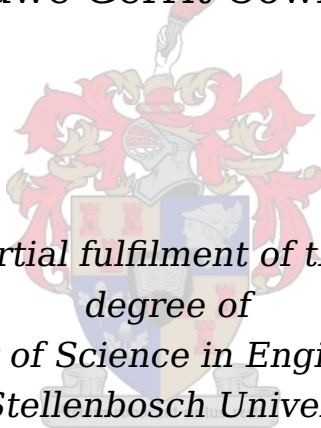


Kinodynamic Planning for a Fixed-Wing Aircraft in
Dynamic, Cluttered Environments:
A Local Planning Method Using Implicitly-Defined
Motion Primitives

by

Edwé Gerrit Cowley



*Thesis presented in partial fulfilment of the requirements for the
degree of
Master of Science in Engineering
at Stellenbosch University*

Supervisor:

Dr Corné Edwin van Daalen

Department Electrical and Electronic Engineering

March 2013

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

March 2013

Abstract

In order to navigate dynamic, cluttered environments safely, fully autonomous Unmanned Aerial Vehicles (UAVs) are required to plan conflict-free trajectories between two states in position-time space efficiently and reliably. Kinodynamic planning for vehicles with non-holonomic dynamic constraints is an NP-hard problem which is usually addressed using sampling-based, probabilistically complete motion planning algorithms. These algorithms are often applied in conjunction with a finite set of simple geometric motion primitives which encapsulate the dynamic constraints of the vehicle. This ensures that composite trajectories generated by the planning algorithm adhere to the vehicle dynamics. For many vehicles, accurate tracking of position-based trajectories is a non-trivial problem which demands complicated control techniques with high energy requirements.

In an effort to reduce control complexity and thus also energy consumption, a generic Local Planning Method (LPM), able to plan trajectories based on implicitly-defined motion primitives, is developed in this project. This allows the planning algorithm to construct trajectories which are based on simulated results of vehicle motion under the control of a rudimentary auto-pilot, as opposed to a more complicated position-tracking system. The LPM abstracts motion primitives in such a way that it may theoretically be made applicable to various vehicles and control systems through simple substitution of the motion primitive set.

The LPM, which is based on a variation of the Levenberg-Marquardt Algorithm (LMA), is integrated into a well-known Probabilistic Roadmap (PRM) kinodynamic planning algorithm which is known to work well in dynamic and cluttered environments. The complete motion planning algorithm is tested thoroughly in various simulated environments, using a vehicle model and controllers which have been previously verified against a real UAV during practical flight tests.

Opsomming

Ten einde dinamiese, voorwerprike omgewings veilig te navigeer, word daar vereis dat volledig-outonome onbemande lugvoertuie konflikvrye trajekte tussen twee posisie-tyd-toestande doeltreffend en betroubaar kan beplan. Kinodinamiese beplanning is 'n NP-moeilike probleem wat gewoonlik deur middel van probabilisties-volledige beplannings-algoritmes aangespreek word. Hierdie algoritmes word dikwels in kombinasie met 'n eindige stel eenvoudige geometriese maneuvres, wat die dinamiese beperkings van die voertuig omvat, ingespan. Sodanig word daar verseker dat trajekte wat deur die beplanningsalgoritme saamgestel is aan die dinamiese beperkings van die voertuig voldoen. Vir baie voertuie, is die akkurate volging van posisie-gebaseerde trajekte 'n nie-triviale probleem wat die gebruik van ingewikkelde, energie-intensiewe beheertegniese vereis.

In 'n poging om beheer-kompleksiteit, en dus energie-verbruik, te verminder, word 'n generiese plaaslike-beplanner voorgestel. Hierdie algoritme stel die groter kinodinamiese beplanner in staat daartoe om trajekte saam te stel wat op empiriese waarnemings van voertuig-trajekte gebaseer is. 'n Eenvoudige beheerstelsel kan dus gebruik word, in teenstelling met die meer ingewikkelde padvolgingsbeheerders wat benodig word om eenvoudige geometriese trajekte akkuraat te volg. Die plaaslike-beplanner abstraeer maneuvres in so 'n mate dat dit teoreties op verskeie voertuie en beheerstelsels van toepassing gemaak kan word deur eenvoudig die maneuver-stel te vervang.

Die plaaslike-beplanner, wat afgelei is van die Levenberg-Marquardt-Algoritme (LMA), word in 'n welbekende "Probabilistic Roadmap" (PRM) kinodinamiese-beplanningsalgoritme geïntegreer. Dit word algemeen aanvaar dat die PRM effektief werk in dinamiese, voorwerprike omgewings. Die volledige beplanningsalgoritme word deeglik in verskeie, gesimuleerde omgewings getoets op 'n voertuig-model en -beheerders wat voorheen vir akkuraatheid teenoor 'n werklike voertuig gekontroleer is tydens praktiese vlugtoetse.

Contents

Abstract	iii
Opsomming	iv
List of Figures	viii
List of Tables	ix
Nomenclature	x
Acknowledgements	xiv
1 Introduction and Problem Description	1
1.1 Background	1
1.2 An Integrated Autonomous Navigation System	2
1.3 Research Objectives	3
1.4 Test Vehicle	4
1.5 Thesis Layout	5
2 Modelling the Vehicle and its Environment	7
2.1 Overview	7
2.2 Definitions	7
2.2.1 Environment Representation	8
2.2.2 Fixed-Wing Aircraft Definitions	8
2.3 Vehicle Model	10
2.4 Point Mass Kinematics	11
2.4.1 Velocity Dynamics	11
2.4.2 Position Dynamics	12
2.4.3 Attitude Dynamics	13
2.5 Modelling Specific Forces and Moments	13
2.5.1 Rotational Dynamics	13
2.5.2 Aerodynamic and Thrust Model	16
2.5.3 Throttle Dynamics	18
2.6 Vehicle Control System	18
2.6.1 Specific Acceleration and Roll Rate Controllers	18
2.6.1.1 Decoupling the Fast Vehicle Dynamics	19
2.6.1.2 Axial Specific Acceleration	20
2.6.1.3 Normal Specific Acceleration	20
2.6.1.4 Lateral Specific Acceleration	20
2.6.2 Point-mass Kinematics Controllers	20

2.6.2.1	Guidance Controller	20
2.6.2.2	Normal Specific Acceleration Vector Direction Controller	21
2.6.2.3	Velocity Controller	21
2.6.3	Controller Summary	21
2.7	Representing System Dynamics for an Autonomous Vehicle	22
2.7.1	Modelling a System by Differential Equations	22
2.7.2	Encapsulating System Dynamics Using Manoeuvres	23
3	A Kinodynamic Planning Algorithm for Dynamic Environments	25
3.1	Overview	25
3.2	Selecting an Algorithm for Motion Planning in Dynamic Environments	25
3.2.1	Combinatorial Algorithms	26
3.2.2	Sampling-based Algorithms	26
3.2.2.1	Incremental Search Algorithms	26
3.2.2.2	Roadmap Algorithms	27
3.3	Motion Planning Algorithm Implementation	29
3.3.1	The Probabilistic Roadmap Method Algorithm	30
3.3.2	Sampling from a Bounded Subset of the State-Time Space	31
3.3.3	Extending the Graph of Reachable Milestones	34
3.3.4	Finding Solutions and Pruning the Graph	35
3.3.5	A Note on Implementation of the Motion Planning Algorithm	36
4	Local Planning In a Cluttered, Dynamic, Three-Dimensional Environment	37
4.1	Overview	37
4.2	Developing Manoeuvres	38
4.2.1	Dubins Curves in Two Spatial Dimensions	38
4.2.2	The Ideal Dubins Airplane	39
4.2.3	Defining Geometrically Perfect Three Dimensional Manoeuvre Sets	39
4.2.4	Practical Implications of Using Geometrically Perfect Manoeuvre Sets	40
4.2.4.1	High Energy Consumption for Geometric Manoeuvres	41
4.2.4.2	High Energy Consumption requirements of Path-Following	43
4.2.5	Practical Approximations to the Ideal Dubins Manoeuvres	44
4.2.5.1	A Simplified Control Approach	44
4.2.5.2	Capturing Nominal Trajectories for Practical Manoeuvres	45
4.3	Local Planning Method	47
4.3.1	Local Path Planning as a Minimisation Problem	48
4.3.2	Selecting an Appropriate Minimisation Algorithm	49
4.3.2.1	Gradient Evaluation Algorithms	50
4.3.2.2	Hessian Evaluation Algorithms	50
4.3.2.3	Hybrid Algorithms	52
4.3.3	Applying Fletcher's Version of the Levenberg-Marquardt Algorithm to the Local Planning Problem	56
4.3.3.1	Convergence	56
4.3.3.2	Computational Complexity	60
4.3.4	Manoeuvre Sequence Selection	61
5	System Evaluation	62
5.1	Overview	62
5.2	Test Environment	62

5.3	Comparing Ideal and Practical Manoeuvre Sets	62
5.3.1	Energy Efficiency	63
5.3.1.1	Quantifying energy consumption	63
5.3.1.2	Comparison of energy consumption using different control approaches	64
5.4	Local Planning Method Evaluation	67
5.4.1	Speed and Reliability	67
5.4.2	Replanning	67
5.5	Kinodynamic Planning Algorithm Evaluation	69
5.5.1	Test Scenario 1: Static Slalom	71
5.5.2	Test Scenario 2: Slowly-Opening Gate	75
5.5.3	Test Scenario 3: Cluttered Dynamic Environment	77
5.5.4	Test Scenario 4: Dynamic Environment With Replanning	81
5.5.5	Per-Function Breakdown of PRM-Algorithm Execution Times	84
6	Conclusion	85
6.1	Summary	85
6.2	Primary Contributions and Notable Results	86
6.3	Future Work	87
A	Simulating the Environment and Conflict Detection	88
B	Estimating the Volume Occupied by a Number of Identical, Uniformly-Distributed and Possibly Overlapping Objects	89
C	Pseudocode for Fletcher's Version of the LMA	91
	Bibliography	93

List of Figures

1.1	Proposed architecture of an integrated autonomous navigation system	2
1.2	Phoenix Trainer 60	5
2.1	Local Tangent Plane (LTP) coordinate system	8
2.2	Body Axis coordinate system and fixed-wing aircraft control planes	9
2.3	Schematic Overview of Vehicle Model.	10
2.4	SAM-based Path Following Control (PFC) system architecture	21
3.1	Construction for determining the spatial axis lengths of the sampling space . .	31
3.2	Example of a sampling space which incorporates a maximum velocity constraint.	33
3.3	Typical PRM graph in a cluttered environment	35
4.1	Relationship between actual vertical velocity and equivalent constant vertical velocity for the Dubins airplane	39
4.2	Example of an ideal Dubins airplane manoeuvre sequence.	41
4.3	Forces acting on a fixed-wing aircraft during a banking turn	42
4.4	Proposed Simple Control (SC) system architecture	45
4.5	Levenberg-Marquardt minimisation traced across several iterations	53
4.6	Update process for the combination coefficient λ in the original Levenberg-Marquardt algorithm	54
4.7	Update process for the combination coefficient λ in Fletcher's version of the Levenberg-Marquardt algorithm	57
4.8	Typical relationship between manoeuvre sequence execution times and the residual-sum-of-squares cost for a turn-straight-turn manoeuvre	58
5.1	Comparison of energy expended for different control systems	66
5.2	Distribution of LPM Execution Times	68
5.3	Comparison of trajectories executed by the simple control system with and without replanning in a windy environment	69
5.4	Sample Solution in Test Scenario 1: Static Slalom	71
5.5	Execution time distribution for Test Scenario 1 with specific goal time	72
5.6	Execution time distribution for Test Scenario 1 with unspecified goal time . . .	73
5.7	Sample solution in Test Scenario 2: Slowly-Opening Gate	75
5.8	Sample solution in Test Scenario 3: Cluttered Dynamic Environment	77
5.9	Relationship between environment clutter factor and PRM-algorithm execution times	79
5.10	Sample solution in Test Scenario 4: Dynamic Environment with Replanning . .	83

List of Tables

4.1	Necessary and sufficient Dubins curve manoeuvre in two spatial dimensions	38
5.1	Comparison of Path Following Control (PFC) and Simple Control (SC) performance for a test environment with wind gusts.	65
5.2	Effect of different number of replanning events using the LPM in an environment with strong wind gusts	70
5.3	Distribution of execution times and trajectory costs for Scenario 1 with specific goal time	72
5.4	Distribution of execution times and trajectory costs for Scenario 1 with unspecified goal time	73
5.5	Distribution of execution times and trajectory costs for Scenario 2 with specific goal time	76
5.6	Distribution of execution times and trajectory costs for Scenario 2 with unspecified goal time	76
5.7	Distribution of execution times and trajectory costs for Scenario 4 replanning stage	81
5.8	Per-Function breakdown of time spent in PRM-Algorithm	84

Nomenclature

Abbreviations and Acronyms

ASA	Axial Specific Acceleration
AVL	Athena Vortex Lattice
BFGS	Broyden-Fletcher-Goldfarb-Shanno method
CGM	Conjugate Gradient Method
CPU	Central Processing Unit
DCM	Direction Cosine Matrix
DLL	Dynamic Link Library
EMF	Electromotive Force
ESC	Electronic Speed Controller
ESL	Electronic Systems Laboratory
FLMA	Fletcher-Levenberg-Marquardt Algorithm
FLOP	Floating Point Operation(s)
FLOPS	Floating Point Operations Per Second
GNA	Gauss Newton Algorithm
GPS	Global Positioning System
IMU	Inertial Measurement Unit
LIDAR	Light Detection And Ranging
LMA	Levenberg-Marquardt Algorithm
LPM	Local Planning Method
NMM	Nelder-Mead Method
NSA	Normal Specific Acceleration
NSAVDC	Normal Specific Acceleration Vector Direction Controller
PFC	Path-Following Control (System)
PRM	Probabilistic Roadmap Method
PRM*	PRM-Star

RAM	Random Access Memory
RMS	Root Mean Square
RPP	Randomised Path Planner
RRT	Rapidly-exploring Random Tree
RRT*	RRT-Star
SAM	Specific Acceleration Matching
SA	Simulated Annealing
SC	Simple Control (System)
SIL	Software-In-the-Loop
SLAM	Simultaneous Localisation And Mapping
LTP	Local Tangent Plane
UAV	Unmanned Aerial Vehicle

Greek Letters

α	Angle of attack
β	Angle of sideslip
γ_c	Clutter factor
θ	Pitch angle
μ	Mean
μ	Manoeuvre sequence (bold)
ρ	Radius
σ	Standard deviation
τ	Execution time of a manoeuvre or engine time constant
Φ	Predicted value
ϕ	Roll angle
ψ	Yaw angle
ω	Angular rate

Uppercase Letters

A_D	Specific acceleration A along reference axis D
F_D	Specific force F along reference axis D
H	Hessian matrix (bold)
I	Identity matrix (bold)

J	Jacobian matrix (bold)
L, M, N	Rolling, Pitching and Yawing Moments
P, Q, R	Angular rates
S	Wing area
T	Thrust
U	Uniform distribution

Lowercase Letters

a	Acceleration
b	Wingspan
g	Gravitational Acceleration
m	Inertial mass
p	Position
q	Dynamic Pressure

Subscripts and Superscripts

0	Initial
B	Body Coordinate System
f	Final
g	Goal
I	Inertial LTP (North East Down) Coordinate System
r	Reduced order
W	Wind Coordinate System

Syntax and Style

x	The scalar x (lowercase)
\mathbf{x}	The vector x (lowercase; bold)
\mathbf{X}	The matrix X (uppercase; bold)
$ \mathbf{x} $	The magnitude of \mathbf{x}
\mathbf{X}^T	The transpose of \mathbf{X}
\mathbf{x}'	Augmented/transformed \mathbf{x}
\mathbf{x}^{AB}	\mathbf{x} in coordinate system A relative to coordinate system B
$\mathbf{x} \Big _A$	\mathbf{x} represented in coordinate system A

$x \sim U(a, b)$ x is distributed uniformly in the range $a \leq x \leq b$

\dot{x} First derivative of x with respect to time (rate of x)

Other Symbols

\bar{c} Mean aerodynamic chord

\mathcal{C} Reduced configuration space

C_D Aerodynamic drag coefficient

C_L Aerodynamic lift coefficient

C_l Aerodynamic roll moment coefficient

C_m Aerodynamic pitch moment coefficient

C_n Aerodynamic yaw moment coefficient

C_y Aerodynamic side force

$\mathcal{C}_{\mathcal{M}}$ Collection of manoeuvres

\mathcal{M} Manoeuvre space

\mathcal{O} Computational time complexity order (“Big-Oh” notation)

\mathcal{S} Sampling space

\bar{V}_a Airspeed

\mathcal{X} Configuration space

Acknowledgements

The author wishes to thank the following people for their contributions to the project.

- Dr. C.E. van Daalen, for his patient guidance and for imparting the necessary knowledge and invaluable insight required in all aspects of this project.
- Mr. J.A.A. Engelbrecht for his role as supporting study leader, notably during its formative stage and in helping to define the project scope.
- The ESL and its sponsors, notably the National Aerospace Centre, for their financial contributions toward the project.
- Sampie Smit, for donating his Phoenix Trainer 60 Simulink model and controllers, which form an integral part of this project.
- Chris Jacquet, for providing a LaTeX template for this thesis and for constant LaTeX and general technical support.
- ESL lab management, notably Lionel Basson and A.M. De Jager, for helping me navigate the ESL code repositories.
- All members of the ESL Autonomous Navigation Research Group, for providing regular feedback and advice throughout the project.

Chapter 1

Introduction and Problem Description

1.1 Background

During the past decade, there has been a massive proliferation of Autonomous Aerial Vehicles (UAVs) for a wide range of applications, including surveillance, search and rescue, relaying communications, creating decoy targets and even active combat. Nearly all UAVs in active service are, however, not fully autonomous, in the sense that they require some degree of operator intervention in order to safely navigate in an environment which is shared with other air traffic.

The main shortcoming which necessitates ground station assistance, is the unreliability and unpredictable performance of automated conflict avoidance systems available for such vehicles, especially in environments with a high number of dynamic obstacles or high dynamic *clutter*.

It must be noted that the concept of conflict differs from that of collision in that conflict does not necessarily imply contact between the vehicle and an obstacle. Conflict is usually defined as a state in which a vehicle is in an unsafe proximity, defined by some exclusion region around the vehicle's centre of gravity, to another object in the environment. Increasing the size of this exclusion region leads to greater safety, but too large an exclusion zone may result in an undesirably high aversion to external objects, leading to reduced manoeuvrability in highly cluttered environments.

Unfortunately the problem of planning conflict-free trajectories has proven to be one that is computationally intractable for a vehicle with non-holonomic dynamic constraints when searching in the entire state space of the vehicle [1; 2]. To this end, many researchers have resorted to an alternate representation of vehicle dynamics through the encapsulation of vehicle dynamics in a finite set of motion primitives or *manoeuvres*. For a fixed-wing aircraft, these manoeuvres are usually approximated by geometrically perfect helices and straight lines in three dimensions. Such trajectories require complex and often energy-intensive path-following control systems which regulate axial accelerations aggressively in order to track geometric position references accurately. However, in dynamic environments, it is often impossible to model objects and their future states in the environment with sufficient certainty to warrant very accurate tracking of a position-based trajectory, due to factors such as sensor noise, model uncertainty and a high uncertainty with regard to the nature and intent of other dynamic objects.

An alternate means of defining manoeuvres, and a reliable motion planning algorithm which is able to use these manoeuvres to construct feasible and unconflicted trajectories

through dynamic, cluttered environments, are therefore required.

1.2 An Integrated Autonomous Navigation System

In order to provide some context to the role of a motion planning algorithm planner, we present a proposed architecture for an integrated autonomous navigation system in Figure 1.1. This structure, adapted from an architecture proposed by Van Daalen [2], shows various modular components of the navigation system as blocks, and the directionality of interaction between these modules are indicated by arrows. This architecture is not specific to a certain vehicle type or class, and is considered equally appropriate for aerial, aquatic and ground-based vehicles.

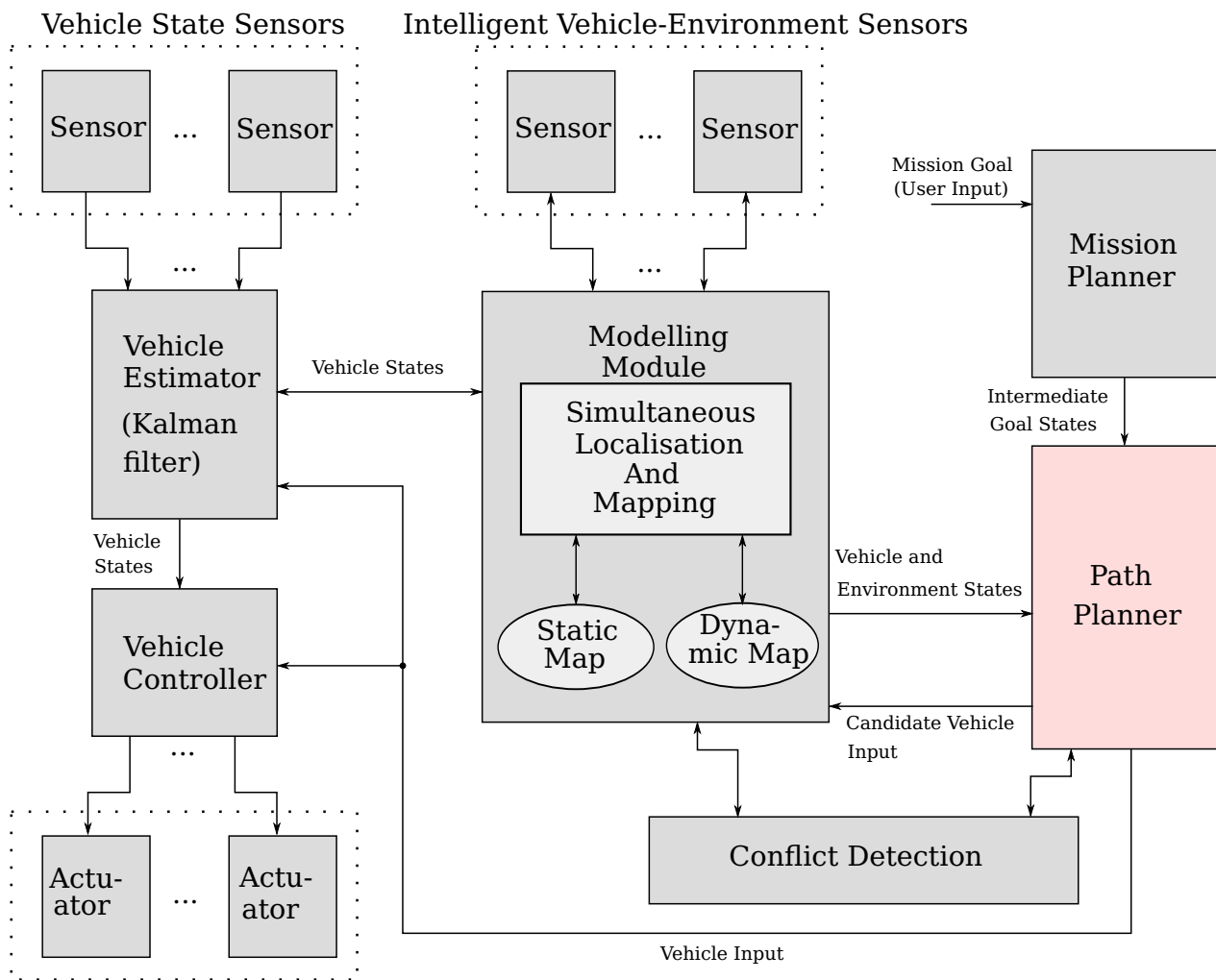


Figure 1.1 – Proposed structure of an integrated autonomous navigation system. Adapted from [2]. The path planner – the focus of this project – is shaded red.

The vertically aligned blocks on the far left of Figure 1.1, including the vehicle state sensors, the Kalman filter estimator, vehicle controller and actuators, all form part of a typical vehicle control system, which is a well-studied topic. The vehicle controller accepts a command, usually in the form of a reference signal in one or more of the vehicle

states, and manipulates the vehicle actuators in order to follow this reference. The vehicle state estimator uses information from an array of standard sensors – including a Global Positioning System (GPS) and Inertial Measurement Unit (IMU) – to estimate the position and orientation, as well as the time derivatives of these states, with respect to an inertial reference frame. Taking drift, sensor noise and model uncertainty into account, the state information generated by the estimator is usually represented probabilistically, by the mean and covariance.

The information provided by the state estimator is combined with data obtained from so-called *intelligent* sensors, which sense vehicle and environment states relative to one another, instead of in relation to an absolute frame of reference. An example of such a sensor is Light Detection And Ranging (LIDAR), which uses light, usually in the form of a laser beam, to sense distances to objects in the environment. Via a process called Simultaneous Localisation And Mapping (SLAM), the information from both the vehicle estimator and intelligent sensors are combined to simultaneously construct a map of the vehicle's environment and to locate the vehicle within that map. A structure containing the estimated states of dynamic objects and static features can typically be maintained by the modelling module.

The path planner developed in this project must plan a trajectory between the initial vehicle state, obtained from the modelling module, to a specified goal state. It requests a conflict analysis by providing a candidate trajectory to the conflict detection module. The conflict module uses data about the vehicle and its environment from the modelling module to estimate the probability for conflict along the trajectory provided. It then returns this probability, which is either evaluated by the path planner to a binary *conflict* or *no conflict* result, or is added to a cumulative conflict estimate for a trajectory consisting of several segments. Once the path planner finds a conflict-free solution trajectory, the corresponding input to execute this trajectory is provided to the vehicle controller.

Tentatively, a higher-level planning module – the mission planner – is proposed. This module takes a long-term mission objective, specified by a human operator, and translates it into a sequence of intermediate goals, which may, for certain applications, be represented by waypoint states. For a UAV, this process may make use of information such as static terrain data, known air traffic routes, restricted and prohibited air zones, air currents, and even weather predictions in order to generate waypoints along favourable routes.

1.3 Research Objectives

As stated in Section 1.1, simple geometric manoeuvre definitions are not very practical for a fixed-wing UAV, and the control systems required to effect the tracking of such trajectories are very aggressive and therefore energy intensive¹.

To this end, we aim to simplify the control system for a fixed-wing aircraft such that it does not control axial accelerations aggressively, and to use this control system to define implicit manoeuvres which represent energy-efficient trajectories for this class of vehicle.

The main focus of the research presented, is the development of a Local Planning Method (LPM) which is able to use these implicit practical manoeuvre definitions to construct

¹This is investigated in Section 4.2.4.

solutions which connect two vehicles states optimally in a conflict-free space. The LPM is contained within the Path Planner functional block shown in Figure 1.1. We shall follow a generic approach in developing this LPM in order to make it applicable not only to fixed-wing UAVs, but to a wide range of vehicles which may have different manoeuvre definitions.

Finally, we aim to incorporate the novel Local Planning Method (LPM) in a larger motion planning algorithm which is able to use the LPM to construct feasible and conflict-free trajectories in dynamic, cluttered environments. We apply the algorithm specifically to the case of a fixed-wing UAV, but, as was already mentioned, develop it using a general approach such that it may be made applicable to a much wider range of vehicles. We set several requirements for this motion planning algorithm, in that it must

1. guarantee *completeness*, i.e. that a solution trajectory will be found provided that at least one solution exists in a given environment,
2. generate only solution trajectories which conform to both the dynamic constraints of the vehicle and the constraints inherent to the environment in which the vehicle operates,
3. ensure that generated trajectories are free of conflict, and
4. be able to plan trajectories in real-time in order to provide alternative solutions should previously planned trajectories become conflicted.

1.4 Test Vehicle

The test vehicle used for this project is a Phoenix Trainer 60 with a Hyperion ZS-4025-10 brushless DC motor. This vehicle has been used in a previous project [3] at the Stellenbosch University Electronic System Laboratory (ESL). A Simulink simulation of the vehicle model and its control system were previously developed [3] based on a model and controllers which have been verified in several other projects undertaken at the ESL. The basic model and controller architectures have been practically flight tested, under both normal and acrobatic flight conditions, on a CAP232 model aerobatic aircraft [4; 5], and as part of an automated take-off and landing control system for the Phoenix Trainer 60 [3] considered in this project, which means that we are confident that the model and controllers are true to reality.

Due to the fact that other modules on which the path planning module relies, such as the conflict detection module and the modelling module (Figure 1.1), have not yet been practically realised and are topics of active research at the ESL, we are only able to test the algorithms we develop using the aforementioned software models. Due to the aforementioned flight tests, we are fairly confident that the models accurately simulate the vehicle and its controllers, and we therefore assume that results obtained using the simulation are a fairly accurate reflection of what can be expected from the real vehicle.

A Specific Acceleration Matching (SAM) control system, proposed by Peddle [7], implemented by Gaum [5] for the CAP232, and adapted for the Phoenix Trainer 60 (Figure 1.2) by Smit [3], is used as a baseline control system for this project. The details of this control system and the changes made to it are provided in Section 4.2.



Figure 1.2 – Phoenix Trainer 60 “Sampioen” Copyright ©2011 ESL Stellenbosch University [6]. Used with permission. The model of this UAV is used to simulate a test vehicle for the algorithms developed in this project.

1.5 Thesis Layout

Chapter 1 introduces the background to the motion planning problem which is addressed in this thesis. The proposed architecture for an integrated autonomous navigation system is presented in order to provide context for the motion planning algorithm. The test vehicle model and controllers are also introduced.

Chapter 2 explains some basic concepts used throughout the document, including definitions of the inertial Local Tangent Plane (LTP) coordinate system and the non-fixed body-axis and wind-axis coordinate systems. The vehicle actuators and control planes, as well as their effects with regard to the various coordinate systems are also discussed. Furthermore, summaries of the vehicle model and controllers are provided. The final part of the chapter introduces two distinct approaches to the representation of vehicle dynamics: the classic state space representation as well as the manoeuvre space representation, which is considered more appropriate for our problem and is used throughout this thesis.

Chapter 3 elaborates on the motion planning problem. It provides a brief literature study of existing motion planning algorithms in the context of dynamic, cluttered environments, and motivates the selection of a specific algorithm (a version of the Probabilistic Roadmap Method). This algorithm, as well as its reliance on a Local Planning Method which is able to plan optimal trajectories between local states in unconflicted space, is explained in detail.

The Local Planning Method (LPM) is discussed in further detail in Chapter 4. The first part of the chapter introduces the geometrically perfect motion primitives which are typically used in other LPMs, and the control systems used to realise these manoeuvres in practice. It is argued that these motion primitives are not representative of the natural pilot manoeuvres for a fixed-wing aircraft, and that the control approach used to effect these manoeuvres is inherently inefficient in terms of energy expended. An alternative, simple approach to the control system and a method of defining implicit manoeuvres from the nominal trajectories generated by this control system, is proposed. In the

second part of Chapter 4, the implementation of an LPM which is able to construct optimal trajectories between initial and goal states in an unconflicted space, is discussed.

In Chapter 5, we provide and analyse data obtained from testing both the LPM as a stand-alone function, and the LPM as integrated into the larger motion planning algorithm, in various environments. These environments are developed such that they test all the requirements set in Section 1.3.

Chapter 6 concludes with the most notable results of the project and the contributions made. Topics for related future research are also proposed.

Chapter 2

Modelling the Vehicle and its Environment

2.1 Overview

This chapter introduces some basic concepts used throughout the document.

The first section defines several coordinate systems used to represent the environment and the state of the vehicle and objects in the environment, both relative to one another and relative to some fixed point in an earth-fixed inertial frame.

The actuators and control planes of a fixed-wing aircraft are also introduced and their effect on the vehicle are discussed. These concepts are applied directly in Section 4.2 in developing manoeuvres, but a basic understanding of the vehicle dynamics and limitations also provides useful insight in other sections.

The second section of this chapter considers alternate representations of the vehicle dynamics: the classic state space representation and the less common manoeuvre space representation. It is argued that the state space representation introduces high time complexity to a motion planning algorithm which must incorporate non-holonomic dynamic constraints, and that the manoeuvre space representation is therefore more suitable for our purposes.

2.2 Definitions

In the development of the motion planning algorithm, the need arises for an inertial reference frame in which to specify position, velocity, acceleration and orientation, both for our vehicle and for other objects in the environment.

Furthermore, we require a means to express the axial forces and accelerations acting on the vehicle body, and a means to convert between this body-axis representation and the inertial reference representation.

To this end, we provide several definitions in this section with regard to coordinate systems for the environment and vehicle. We also discuss the effect of the actuators and control planes of a fixed-wing aircraft in terms of the accelerations that they induce around or along the axes in the various coordinate systems.

2.2.1 Environment Representation

In order to describe the environment and the position, position derivatives and orientation of the vehicle and various obstacles in the environment, we use coordinates on the Local Tangent Plane (LTP) reference axes. This coordinate system defines a flat plane which is locally tangential to the surface of the Earth, and which has axes aligned with true North and East. The third orthogonal axis points down, towards the centre of the Earth, in compliance with the definition of a right-handed system of axes. The LTP axes represent an inertial coordinate system which ignores curvature and rotation of the Earth. This approximation is considered adequate for local navigation.

Figure 2.1 illustrates the LTP coordinate system definition. The X_I , Y_I and Z_I inertial axes are shown to align with the North East and Down axes respectively.

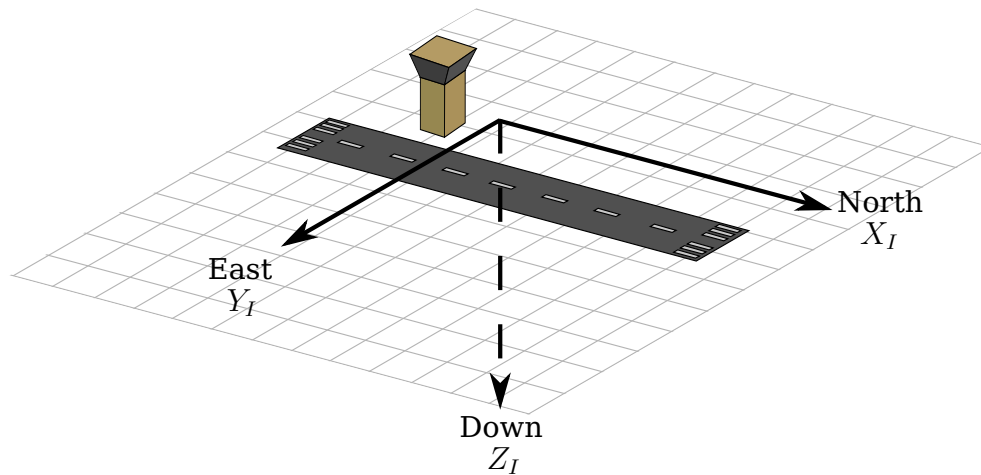


Figure 2.1 – Local Tangent Plane (LTP) coordinate system.

The origin of the LTP coordinate system is chosen as a fixed reference on the ground, often the original location of the aircraft on the runway before departure.

2.2.2 Fixed-Wing Aircraft Definitions

Forces and moments generated by the aircraft engine and the various control planes are defined in a coordinate system which translates and rotates with the vehicle. This is the body-coordinate system.

The origin of this system of axes is the centre of gravity of the vehicle. The X_B (body) axis points directly ahead of the aircraft along its line of symmetry, the Y_B axis points along the starboard wing and the Z_B axis points down, with the three axes being mutually orthogonal.

The body-axis system is shown in Figure 2.2. In this figure, the control planes of the aircraft are highlighted in different colours. A differential deflection of the ailerons, induced by a command to the aileron actuators, causes a rolling moment about the X body axis, an elevator command causes a pitching moment about the Y body axis, and a rudder command causes a yawing moment about the Z_B axis. The positive direction of each moment is indicated by an arrow on the corresponding axis in Figure 2.2. A throttle command induces acceleration along the positive X_B axis.

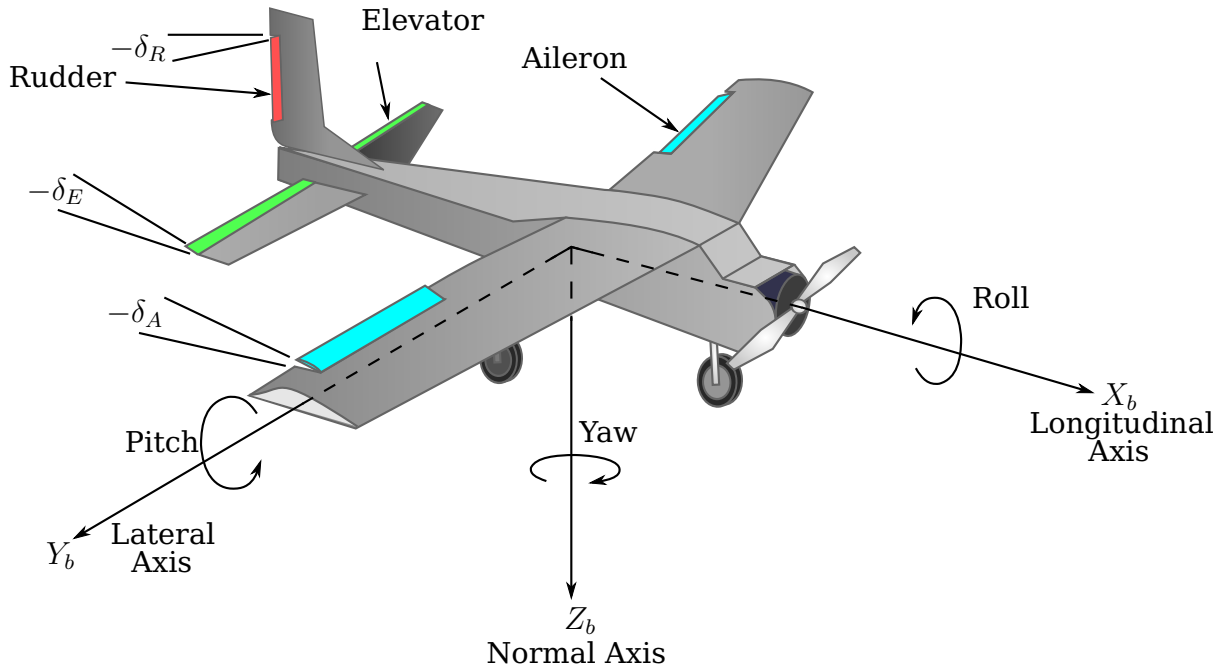


Figure 2.2 – Body Axis coordinate system and fixed-wing aircraft control planes. The ailerons are indicated in aqua, the elevator, in green, and the rudder, in red.

The simulated aircraft model and control system use the body axis coordinate system to represent the forces and moments generated by the vehicle, but express the vehicle position, velocity, acceleration and orientation in the LTP coordinate system [5], because the body axes do not represent an inertial coordinate system. The Coriolis equation is used to transform vectors between the body coordinate system and the LTP coordinate system [5]. The motion planning algorithm which we develop operates exclusively in the LTP coordinate system and interfaces with the vehicle controllers only using vectors expressed in terms of the North, East and Down axes.

A third, right-handed system of axis, which shares its origin with the body axis system, is oriented so that its X -axis is always directed along the aircraft's velocity vector (i.e. opposing the oncoming "wind" vector), with its Z -axis directed downward in the plane of symmetry of the vehicle. This defines the wind coordinate system. The angle between the X_B (body) and X_W (wind) axes is called the angle of attack α , and the angle between the Y_B and Y_W axes is called the angle of sideslip β .

We use the term *trim flight* to describe a state in which the angle of attack and airspeed are such that the aircraft maintains constant altitude, with the additional implication that both roll and yaw rates are zero.

2.3 Vehicle Model

As a test environment for the algorithm which we develop, an existing simulation of the Phoenix Trainer 60 introduced in Section 1.4 is adapted for our purposes. This software in the loop (SIL) Simulink environment, as well as the vehicle model which it uses, was developed by Smit [3], based on previous work by Peddle [7] and other researchers at the ESL. Although the model and controllers are used essentially as black box tools during this project, save for a few minor changes to the outer loop controllers (refer to Section 4.2.5.1), an overview of the vehicle model and controllers provide valuable insight into the dynamic constraints on the vehicle. The research presented in this section relates to the vehicle dynamics and vehicle model, and is condensed mainly from publications by Peddle [7], Gaum [5] and Smit [3].

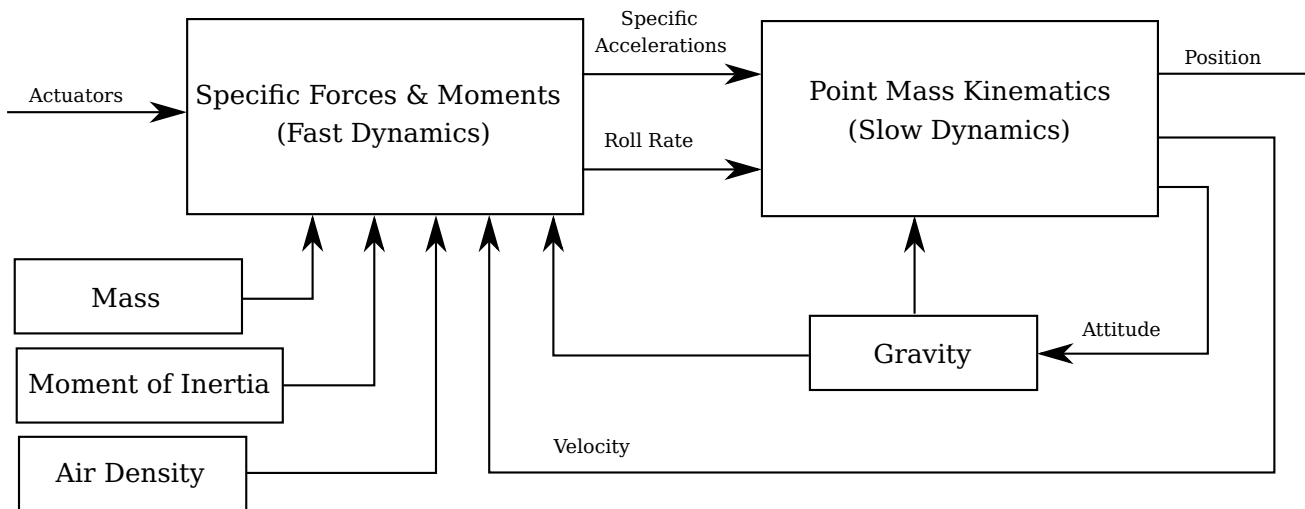


Figure 2.3 – Schematic Overview of Vehicle Model [7].

The vehicle model is shown schematically in Figure 2.3. The model is split into two distinct parts: the point-mass kinematics model, and a model of specific forces and moments [7; 5]. The former models the vehicle as a point-mass with commandable specific accelerations and roll rate. This typically represents the slower dynamics of the aircraft, and is applicable to any fixed-wing aircraft. The latter model represents the faster dynamics of the aircraft, and is unique for every specific aircraft. The distinction between faster and slower dynamics allows a similar separation for the vehicle controllers, as will be explained later in this chapter. The following subsections provide additional insight into the vehicle model.

2.4 Point Mass Kinematics

Based on the work of Peddle [7], Smit [3] derived a model for the vehicle which essentially reduces the entire system to a point mass with commandable specific axial accelerations and associated roll rate. This model encapsulates the slower dynamics of the aircraft. The control inputs to the point mass kinematics model are specific accelerations along the three inertial axes and a corresponding roll rate.

2.4.1 Velocity Dynamics

In this subsection, we express vehicle velocity (and airspeed) in terms of specific accelerations of the vehicle along the wind axes.

The total acceleration vector and gravity vector are related to the velocity vector by [7; 5; 3]

$$\left. \frac{d}{dt} \mathbf{v}^{WI} \right|_I = \mathbf{a}^{WI} + \mathbf{g}. \quad (2.4.1)$$

where

\mathbf{a} represents total acceleration,
 \mathbf{v} represents velocity, and
 \mathbf{g} represents acceleration due to gravity.

In the above equation, the superscripts denote relative values between coordinate systems. In this particular case, the WI superscript denotes that the vector is in the wind coordinate system relative to the inertial coordinate system. A subscript denotes that a vector is represented in (has been transformed to) a particular coordinate system.

Using the Coriolis equation, the velocity time-derivative vector may be transformed from the inertial axis system to a system of axes which rotates with the vehicle [5]:

$$\left. \frac{d}{dt} \mathbf{v}^{WI} \right|_W = -\boldsymbol{\omega}^{WI} \times \mathbf{v}^{WI} + \mathbf{a}^{WI} + \mathbf{g}, \quad (2.4.2)$$

where

$\boldsymbol{\omega}$ is a vector which denotes the rate of angular rotation.

Applying a transformation to simplify the cross product and using the Direction Cosine Matrix (DCM) to coordinate the gravity vector in the wind axes, Equation 2.4.2 may be written as

$$\begin{bmatrix} \dot{\|\mathbf{v}\|} \\ 0 \\ 0 \end{bmatrix} = - \begin{bmatrix} 0 & -R_W & Q_W \\ R_W & 0 & -P_W \\ -Q_W & P_W & 0 \end{bmatrix} \begin{bmatrix} \|\mathbf{v}\| \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} A_{Wx} \\ A_{Wy} \\ A_{Wz} \end{bmatrix} + \mathbf{DCM}^{WI} \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix}, \quad (2.4.3)$$

where

P_W , Q_W , and R_W are angular rates about the wind axes,
 A_{Wx} , A_{Wy} , and A_{Wz} are specific accelerations along the wind axes,
 \mathbf{DCM} is the direction cosine matrix, and
 g is acceleration due to gravity.

From Equation 2.4.3, the velocity derivative magnitude may be expressed as [5]

$$||\dot{\mathbf{v}}|| = A_{Wx} + ge_{13}^{WI}, \quad (2.4.4)$$

where

e_{13}^{WI} represents row 1 and column 3 of the **DCM**^{WI}, as per the notation of Gaum [5].

Extracting the two remaining constituent equations from Equation 2.4.3 and applying Newton's second law, we obtain two constraints [5]:

$$R_W = \frac{1}{m||\mathbf{v}||} F_{W_Y}, \quad (2.4.5)$$

and

$$Q_W = -\frac{1}{m||\mathbf{v}||} F_{W_Z}, \quad (2.4.6)$$

where

m is the inertial mass of the vehicle,
 F_{W_Y}, F_{W_Z} are force in the Y and Z wind axes respectively and,
 \mathbf{v} is velocity in the wind axes.

2.4.2 Position Dynamics

Proceeding as per the previous section, we may express the position dynamics for the vehicle in terms of the velocity dynamics:

$$\frac{d}{dt} \mathbf{p}^{WI} \Big|_I = \mathbf{v}^{WI} \quad (2.4.7)$$

where

\mathbf{p} represents position, and
 \mathbf{v} represents velocity.

It is possible to transform to the inertial coordinate system using the DCM. Knowing that the DCM is an orthogonal matrix, we can write [5]

$$\dot{\mathbf{p}}_I^{WI} = [\mathbf{DCM}^{WI}]^T \mathbf{v}_W^{WI}. \quad (2.4.8)$$

Equation 2.4.8 can be expanded as [3]

$$\begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{p}_z \end{bmatrix} = \begin{bmatrix} \cos \psi_W \cos \theta_W \\ \sin \psi_W \cos \theta_W \\ -\sin \theta_W \end{bmatrix} ||\mathbf{v}||. \quad (2.4.9)$$

2.4.3 Attitude Dynamics

Using the Euler-3-2-1 representation of vehicle orientation, the attitude dynamics for a fixed-wing aircraft can be written as [8]

$$\begin{bmatrix} \dot{\phi}_W \\ \dot{\theta}_W \\ \dot{\psi}_W \end{bmatrix} = \begin{bmatrix} 1 & \sin \phi_W \tan \theta_W & \cos \phi_W \tan \theta_W \\ 0 & \cos \phi_W & -\sin \phi_W \\ 0 & \sin \phi_W \sec \theta_W & \cos \phi_W \sec \theta_W \end{bmatrix}_{321} \begin{bmatrix} P_W \\ Q_W \\ R_W \end{bmatrix} \quad (2.4.10)$$

The proof for this result, taken from the research of Peddle [8], is omitted for the sake of brevity.

2.5 Modelling Specific Forces and Moments

The fast vehicle dynamics are captured by a model which describes the forces and moments acting on the vehicle. This section provided a summary of these moments and forces. This model, developed by Peddle [7], has been applied in various ESL projects [5; 3].

2.5.1 Rotational Dynamics

The fixed-wing aircraft is considered a rigid body. Rotational moments about the body axes arise due to relative angular offsets between the body axes and the corresponding wind axes. These offsets are described by the angle of attack and angle of sideslip, as defined in Section 2.2.2.

Relative angular velocity between the wind and inertial axes is given by the sum of the relative angular velocity between the wind and body axes and of that between the body and inertial axes; that is to say [7; 5]

$$\omega^{WI} = \omega^{WB} + \omega^{BI}. \quad (2.5.1)$$

Rewriting Equation 2.5.1 in terms of the angle of attack and angle of sideslip, we obtain the form [7; 5]

$$\omega_B^{BI} = \dot{\alpha} \mathbf{j}_B^B - \dot{\beta} \mathbf{k}_B^W + \omega_B^{WI}, \quad (2.5.2)$$

where

\mathbf{j} is a unit vector along the y axis, and
 \mathbf{k} is a unit vector along the z axis.

with all vectors coordinated in the body axes. Using a rotation matrix, this equation can be rewritten with all vectors in their respective original coordinate systems [5]:

$$\omega_B^{BI} = \dot{\alpha} \mathbf{j}_B^B - \dot{\beta} \mathbf{DCM}^{BW} \mathbf{k}_W^W + \mathbf{DCM}^{BW} \omega_W^{WI}. \quad (2.5.3)$$

Equation 2.5.3 may be written in matrix form as [5]

$$\begin{bmatrix} P \\ Q \\ R \end{bmatrix} \begin{bmatrix} 0 & \sin \alpha \\ 1 & 0 \\ 0 & -\cos \alpha \end{bmatrix} \begin{bmatrix} \dot{\alpha} \\ \dot{\beta} \end{bmatrix} + \begin{bmatrix} \cos \alpha \cos \beta & -\cos \alpha \sin \beta & -\sin \alpha \\ \sin \beta & \cos \beta & 0 \\ \sin \alpha \cos \beta & -\sin \alpha \sin \beta & \cos \alpha \end{bmatrix} \begin{bmatrix} P_W \\ Q_W \\ R_W \end{bmatrix} \quad (2.5.4)$$

Rearranging the above equation and substituting the two constraints from Equations 2.4.5 and 2.4.6, we obtain

$$\begin{bmatrix} \dot{\alpha} \\ \dot{\beta} \\ P_W \end{bmatrix} = \begin{bmatrix} -\cos \alpha \tan \beta & 1 & -\sin \alpha \tan \beta \\ \sin \alpha & 0 & -\cos \alpha \\ \cos \alpha \sec \beta & 0 & \sin \alpha \sec \beta \end{bmatrix} \begin{bmatrix} P \\ Q \\ R \end{bmatrix} + \frac{1}{m||\mathbf{v}||} \begin{bmatrix} \sec \beta & 0 \\ 0 & 1 \\ -\tan \beta & 0 \end{bmatrix} \begin{bmatrix} F_{W_Z} \\ F_{W_Y} \end{bmatrix} \quad (2.5.5)$$

The matrix form of Equation 2.5.5 encapsulates three separate algebraic equations. The first two express the attitude dynamics, represented by the angle of attack and angle of sideslip [5]. The remaining equation constrains the wind axes to be normal to the vehicle's plane of symmetry [5].

We know from Euler's law for rigid bodies that the angular momentum (\mathbf{h}) is related to the applied moment \mathbf{m} by

$$\mathbf{m} = \frac{d}{dt} \mathbf{h} \Big|_I. \quad (2.5.6)$$

Using the Coriolis equation, we may rewrite the above equation in terms of the body axes as [3; 5]

$$\mathbf{m} = \frac{d}{dt} \mathbf{h} \Big|_B + \boldsymbol{\omega}^{BI} \times \mathbf{h}. \quad (2.5.7)$$

Furthermore, the angular momentum in the body axis system is known to be [9]

$$\mathbf{h}_B = \mathbf{R}_B \boldsymbol{\omega}_B^{BI}, \quad (2.5.8)$$

where

\mathbf{R} is the moment of inertia matrix for the vehicle.

Substituting Equation 2.5.8 into Equation 2.5.7 and converting to body axes, we find that [3; 5]

$$\dot{\boldsymbol{\omega}}_B^{BI} = \mathbf{R}_B^{-1} (\mathbf{m}_B - \mathbf{S}_B^{BI} \mathbf{R}_B \boldsymbol{\omega}_B^{BI}). \quad (2.5.9)$$

In the previous equation, the skew matrix \mathbf{S} is given by [5]

$$\mathbf{S} = \begin{bmatrix} 0 & -F_{Bz} & F_{By} \\ F_{Bz} & 0 & -F_{Bx} \\ -F_{By} & F_{Bx} & 0 \end{bmatrix}. \quad (2.5.10)$$

Substituting Equation 2.5.5 into Equation 2.5.9, we finally obtain equations describing the rotational dynamics for the fixed-wing aircraft model completely [7; 5; 3]:

$$\begin{bmatrix} \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -\cos \alpha \tan \beta & 1 & -\sin \alpha \tan \beta \\ \sin \alpha & 0 & -\cos \alpha \end{bmatrix} \begin{bmatrix} P \\ Q \\ R \end{bmatrix} + \frac{1}{m||\mathbf{v}||} \begin{bmatrix} \sec \beta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} F_{Wz} \\ F_{Wy} \end{bmatrix} \quad (2.5.11)$$

and

$$\begin{bmatrix} \dot{P} \\ \dot{Q} \\ \dot{R} \end{bmatrix} = \mathbf{R}_B^{-1} \left(\begin{bmatrix} L \\ M \\ N \end{bmatrix} - \begin{bmatrix} 0 & -R & Q \\ R & 0 & -P \\ -Q & P & 0 \end{bmatrix} \mathbf{R}_B \begin{bmatrix} P \\ Q \\ R \end{bmatrix} \right), \quad (2.5.12)$$

subject to

$$P_W = \begin{bmatrix} \cos \alpha \sec \beta & 0 & \sin \alpha \sec \beta \end{bmatrix} \begin{bmatrix} P \\ Q \\ R \end{bmatrix} + \frac{1}{m||\mathbf{v}||} \begin{bmatrix} -\tan \beta & 0 \end{bmatrix} \begin{bmatrix} F_{Wz} \\ F_{Wy} \end{bmatrix}. \quad (2.5.13)$$

2.5.2 Aerodynamic and Thrust Model

The thrust produced by the propulsion system of the vehicle, together with aerodynamic forces generated by the body, define the *specific forces and moments* acting on the vehicle [5]. The model used to describe these forces and moments acting on our vehicle, is valid under the following conditions [5]:

- The thrust vector is along the X-body axis and does not cause significant moments.
- Main wing downwash on the horizontal tail plane is negligible.
- The vehicle is not operating near stall conditions.

The final point above implies that incidence angles are small. The model which applies to these conditions – the small incidence angle aerodynamic model – is summarised by the following equations [10; 7; 5; 3]:

$$\begin{bmatrix} F_{W_X} \\ F_{W_Y} \\ F_{W_Z} \end{bmatrix} = qS \begin{bmatrix} -C_D \\ C_y \\ -C_L \end{bmatrix} + \begin{bmatrix} \cos \alpha \cos \beta \\ -\cos \alpha \sin \beta \\ -\sin \alpha \end{bmatrix} T \quad (2.5.14)$$

$$\begin{bmatrix} L_W \\ M_W \\ N_W \end{bmatrix} = qS \begin{bmatrix} b & 0 & 0 \\ 0 & \bar{c} & 0 \\ 0 & 0 & b \end{bmatrix} \begin{bmatrix} C_l \\ C_m \\ C_n \end{bmatrix} \quad (2.5.15)$$

, where

T is thrust magnitude,

q is dynamic pressure,

S is the wing area,

C_L is the coefficient of aerodynamic lift,

C_y is the coefficient of aerodynamic side force,

C_D is the coefficient of aerodynamic drag,

b is the wing span, and

\bar{c} is the mean aerodynamic chord.

The dynamic pressure is [5]

$$q = \frac{1}{2} \rho ||\mathbf{v}||^2, \quad (2.5.16)$$

where

ρ is the air density.

The various aerodynamic coefficients in Equations 2.5.14 and 2.5.15 are given by [10; 5]:

$$C_D = C_{D_0} + \frac{C_L^2}{\pi A e}, \quad (2.5.17)$$

$$\begin{bmatrix} C_y \\ C_L \end{bmatrix} = \begin{bmatrix} 0 \\ C_{L_0} \end{bmatrix} + \begin{bmatrix} 0 & C_{y_\beta} & \frac{b}{2\bar{V}_a} C_{y_P} & 0 & \frac{b}{2\bar{V}_a} C_{y_R} \\ C_{L_\alpha} & 0 & 0 & \frac{\bar{c}}{2\bar{V}_a} C_{L_Q} & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ P \\ Q \\ R \end{bmatrix} + \begin{bmatrix} C_{y_{\delta_A}} & 0 & C_{y_{\delta_R}} \\ 0 & C_{y_{\delta_E}} & 0 \end{bmatrix} \begin{bmatrix} \delta_A \\ \delta_E \\ \delta_R \end{bmatrix}, \quad (2.5.18)$$

and

$$\begin{bmatrix} C_l \\ C_m \\ C_n \end{bmatrix} = \begin{bmatrix} 0 \\ C_{m_0} \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & C_{l_\beta} & \frac{b}{2\bar{V}_a} C_{l_P} & 0 & \frac{b}{2\bar{V}_a} C_{l_R} \\ C_{m_\alpha} & 0 & 0 & \frac{\bar{c}}{2\bar{V}_a} C_{m_Q} & 0 \\ 0 & C_{n_\beta} & \frac{b}{2\bar{V}_a} C_{n_P} & 0 & \frac{b}{2\bar{V}_a} C_{n_R} \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ P \\ Q \\ R \end{bmatrix} + \begin{bmatrix} C_{l_{\delta_A}} & 0 & C_{l_{\delta_R}} \\ 0 & C_{m_{\delta_E}} & 0 \\ C_{n_{\delta_A}} & 0 & C_{n_{\delta_R}} \end{bmatrix} \begin{bmatrix} \delta_A \\ \delta_E \\ \delta_R \end{bmatrix}, \quad (2.5.19)$$

where

- \bar{V}_a is the airspeed
- A is the wing aspect ratio,
- e is the Oswald efficiency factor,
- C_{D_0} is the parasitic drag coefficient,
- C_{L_0} is the static lifting moment coefficient,
- C_{m_0} is the static pitching moment coefficient, and
- terms of the form C_{A_x} are stability and control derivatives.

In the above equation, the stability and control derivatives are expressed as

$$C_{A_x} = \frac{\partial C_A}{\partial x'}, \quad (2.5.20)$$

with

$$x' = n_c x. \quad (2.5.21)$$

The normalising coefficient n_c for pitch rate is

$$n_{c_{\text{pitch}}} = \frac{\bar{c}}{2\bar{V}_a}, \quad (2.5.22)$$

and for roll rate and yaw rate it is given by

$$n_{c_{\text{roll/yaw}}} = \frac{b}{2\bar{V}_a}. \quad (2.5.23)$$

The stability and control derivatives for our model were determined by Smit [3] using *Athena Vortex Lattice*, a tool developed by MIT.

It is important to adhere to the aforementioned conditions in order for our model to be valid. If these restrictions are not enforced, the model ceases to be an accurate description of the actual vehicle, meaning that any results obtained using the model under conditions which violate these restrictions are invalid.

2.5.3 Throttle Dynamics

The vehicle used in this project is propelled by a single Hyperion ZS-4025-10 brushless DC motor, which has an extremely fast step response [3]. The response is modelled by the first order differential equation [3]

$$\frac{d}{dt}T = -\frac{1}{\tau_T}T + \frac{1}{\tau_T}T_C, \quad (2.5.24)$$

where

τ_T is a time constant, and

T_C is the thrust command.

The effect of the velocity magnitude is ignored [3] due to the fact that its effect is negligible [3; 5].

2.6 Vehicle Control System

Having defined the vehicle dynamics using two distinct, cooperative models, a control system which regulates both the fast dynamics, as well as the slower point-mass kinematics, of the vehicle, is introduced. A Specific Acceleration Matching (SAM) control system for the vehicle used in this project was available from the outset, due to the efforts of Smit [3], based on the research of Peddle [7] and Gaum [5].

The control system was originally implemented in Simulink by Gaum [5] for a different aircraft, and was subsequently adapted by Smit [3] for the Phoenix Trainer 60. This section provides a brief overview of the original control system, which tracks position-based reference trajectories aggressively. An adapted version of the control system, which regulates the vehicle's course less strictly, is proposed in Chapter 4. The adapted control system is more susceptible to disturbances (e.g. wind), and some responsibility for coping with these disturbances is therefore delegated to the path planning algorithm. This serves as a means of reducing overall energy expenditure. The specific benefits presented by this configuration are discussed in Chapter 4.

2.6.1 Specific Acceleration and Roll Rate Controllers

As was shown in the preceding sections, the vehicle model may be separated into two distinct parts: a fast, vehicle-specific forces-and-moments model and a model which represents slower point-mass kinematics of the vehicle. In this subsection, we provide a brief summary of a technique introduced by Peddle [7] by which the former model may be decoupled further, into three independent axial accelerations and corresponding roll

rate. We also provide an overview of the separate controllers devised for this decoupled system. These controllers were used as a black-box tool for the purposes of this project, and a detailed design of the controllers is therefore not included. A basic knowledge of their function is, however, necessary to understand the changes to the control system proposed in Chapter 4.

2.6.1.1 Decoupling the Fast Vehicle Dynamics

In order to decouple the fast vehicle dynamics, Peddle [7] introduced several simplifications. Firstly, the cross coupling in Equation 2.5.12 is ignored, as the cross-coupling terms are negligible unless there are significant concurrent angular rates about two axes [7]. This is considered uncommon, even during aggressive manoeuvring [5]. Subject to this simplification, Equations 2.5.12 and 2.5.11 become [7; 5]

$$\begin{bmatrix} \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -\cos \alpha \tan \beta & 1 & -\sin \alpha \tan \beta \\ \sin \alpha & 0 & -\cos \alpha \end{bmatrix} \begin{bmatrix} P \\ Q \\ R \end{bmatrix} + \frac{1}{m||\mathbf{v}||} \begin{bmatrix} \sec \beta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} F_{W_Z} \\ F_{W_Y} \end{bmatrix} \quad (2.6.1)$$

and

$$\begin{bmatrix} \dot{P} \\ \dot{Q} \\ \dot{R} \end{bmatrix} = \begin{bmatrix} \frac{1}{R_X} & 0 & 0 \\ 0 & \frac{1}{R_Y} & 0 \\ 0 & 0 & \frac{1}{R_Z} \end{bmatrix} \begin{bmatrix} L \\ M \\ N \end{bmatrix}. \quad (2.6.2)$$

where

L, M, N are rolling, pitching and yawing moments,
 P, Q, R are the corresponding angular rates, and
 R_X, R_Y, R_Z are the components of R about the subscripted axes.

Additionally, the following approximations are introduced [7; 5]:

- The static lifting moment C_{L_0} and static pitching moment C_{m_0} are taken to be zero.
- Small angle approximations for the angle of attack α and angle of sideslip β are made.
- Thrust is only considered to affect axial dynamics and not normal or lateral dynamics. This approximation is valid considering the relatively low bandwidth of throttle dynamics relative to the high bandwidth lateral and normal controllers.

Using these assumptions, the axial, normal and lateral dynamics can be separated, and independent controllers can be designed. The following sections provide brief qualitative summaries of the axial, normal and lateral decoupled models and their respective acceleration controllers. The detailed design of these controllers falls outside the scope of this project, and can be found in the work of Peddle [7], Gaum [5] and Smit [3].

2.6.1.2 Axial Specific Acceleration

The Axial Specific Acceleration (ASA) controller is the lowest-bandwidth controller of the three controllers which regulate the fast dynamics of the aircraft. This controller provides a throttle command to regulate the acceleration along the relevant axis. Aerodynamic drag is modelled as a disturbance signal and the model is augmented with an integrator to reject it. As the ASA controller is very slow, it can not be time-scale decoupled from the point kinematics controllers. For this reason, the ASA dynamics must be taken into account in the point-mass kinematics controllers [7; 5].

2.6.1.3 Normal Specific Acceleration

Normal acceleration is decoupled from gravitational acceleration in order to develop a Normal Specific Acceleration (NSA) controller which does not depend on the vehicle attitude. This is achieved through direct feedback linearisation, which is detailed in research by Peddle [7].

2.6.1.4 Lateral Specific Acceleration

Lateral Specific Acceleration (LSA) is regulated to zero using the rudder, since lateral motion is undesirable during most fixed-wing aircraft manoeuvres. A second control system, used to damp the vehicle's Dutch Roll mode, is allowed to actuate the rudder cooperatively with the LSA controller. These controllers do not interfere, as the Dutch Roll damping has only frequency components which are significantly higher than the LSA controller [7; 5].

2.6.2 Point-mass Kinematics Controllers

The controllers for the fast, vehicle-specific dynamics of the aircraft (summarised in the preceding section) essentially reduce the aircraft to a point mass with commandable specific accelerations and roll rate. Encapsulating the fast dynamics thus, eliminates attitude dependence [5]. This allows us to define a generic set of time-scale decoupled controllers to regulate the slower point-mass kinematics of the vehicle.

A technique introduced by Peddle [7] has proven to be effective for trajectory-tracking by real-world aircraft [5]. This control approach, termed Specific Acceleration Matching (SAM), assumes that the fast-dynamics controllers, as presented in the previous section, are able to effect instant normal and axial accelerations [7; 5].

The control architecture comprises a guidance controller, a Normal Specific Acceleration Vector Direction Controller (NSAVDC) and a velocity controller. Changes to these lower-bandwidth controllers are proposed in Chapter 4, but for the initial phase of the project, the controllers are used as defined by Peddle [7] and implemented by Gaum [5] and Smit [3]. A summary of the most important aspects of the SAM architecture is provided in this section.

2.6.2.1 Guidance Controller

The guidance controller regulates specific accelerations of the vehicle such that they counteract position and velocity errors, which are defined relative to a position-reference trajectory [7; 5]. A specific acceleration vector is calculated and then separated into NSA

and ASA components, which are regulated by the respective controllers responsible for these accelerations. The specific implementation of SAM used in this project does not regulate velocity within the guidance controller and this controller therefore typically generates small ASA commands.

2.6.2.2 Normal Specific Acceleration Vector Direction Controller

Based on the NSA command generated by the guidance controller, the Normal Specific Acceleration Vector Direction Controller (NSAVDC) generates a roll rate command to effect the correct orientation to achieve the specified NSA. By approximation, the transition to any orientation is considered to be instantaneous [7; 5].

2.6.2.3 Velocity Controller

The velocity controller provides a throttle command to regulate velocity independently from the guidance controller. In order for the velocity controller to function in conjunction with the guidance controller, a predictive method of “kinetic energy analysis with logic switching” is used in order to ensure an appropriate, preemptive response based on the known future trajectory tracked by the guidance controller [5].

2.6.3 Controller Summary

The slower point-mass kinematics controllers, together with the faster, vehicle-specific controllers which regulate specific accelerations and roll rate, are represented schematically in Figure 2.4. Specific accelerations and roll rate commands generated by the point-mass kinematics controllers based on position-reference errors, are fed to the fast, inner loop controllers, which encapsulate the dynamics of the aircraft and essentially reduce the vehicle to a point mass with commandable specific acceleration and roll rate [5]. Control-plane deflections and throttle commands are generated by the faster vehicle-specific controllers.

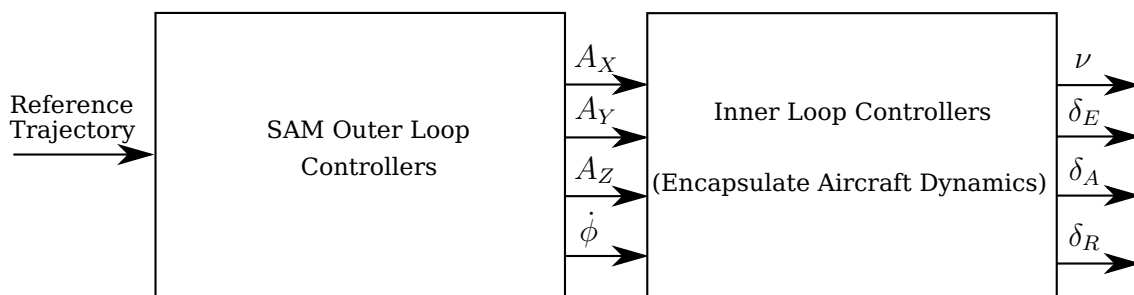


Figure 2.4 – SAM-based Path Following Control (PFC) system architecture [7; 5]. Feedback paths are not shown, but may be inferred from Figure 2.3

2.7 Representing System Dynamics for an Autonomous Vehicle

A kinodynamic planning algorithm requires an accurate model of the vehicle for which it is developed such that it is able to generate trajectories which conform to all the dynamic constraints of the vehicle. This requires a method by which to encapsulate the vehicle dynamics accurately, and in a way which does not impede the ability of the planning algorithm to execute in real-time.

To this end, we shall first consider the common state space representation of vehicle dynamics. We argue that state space representation is not suitable for the motion planning problem due to the high time complexity associated with the large number of vehicle states required to represent the dynamics of a vehicle with non-holonomic constraints.

We subsequently introduce the manoeuvre space representation as an alternative, which allows the encapsulation of several dynamic constraints in a small, finite set of motion primitive trajectories. This leads to a greatly reduced number of search parameters for the motion planning algorithm.

2.7.1 Modelling a System by Differential Equations

The *state space* representation is a standard means of describing the dynamics of an autonomous vehicle. This representation captures the input and output of the vehicle as ordinary differential equations of the form [1; 11; 2],

$$\dot{\mathbf{x}} = f(\mathbf{x}(t), \mathbf{u}(t)), \quad (2.7.1)$$

with

$$\begin{aligned} \mathbf{x} &\in \mathcal{X} \text{ and} \\ \mathbf{u} &\in \mathcal{U}, \end{aligned}$$

where

- x is the state,
- u is the control input,
- \mathcal{X} is the state space, an n -dimensional manifold, and
- \mathcal{U} is the m -dimensional control input space.

Additionally, inequality constraints may be specified in the form [11]

$$F(\mathbf{x}(t), \mathbf{u}(t)) \leq \mathbf{0}, \quad (2.7.2)$$

where F represents a vector of constraints to which the inequality is applied element-wise.

Take into consideration that we only wish to specify a certain subset of the complete state space. For example, with regard to the problem of motion planning for a fixed-wing UAV, we may wish to specify only position and heading at the goal state, and are not concerned about other variables in \mathcal{X} , provided that they are within bounds for safe operation of the vehicle.

To this end, we define a *reduced configuration space* [1; 11]

$$\mathcal{C} \subset \mathcal{X}. \quad (2.7.3)$$

The variables in \mathcal{C} can be considered those variables in which the dynamics of the vehicle are invariant. It can be shown that the state space can be expressed locally as [11]

$$\mathcal{X} = \mathcal{C} \times \mathcal{Y}, \quad (2.7.4)$$

where \mathcal{Y} captures the state variables not contained in \mathcal{C} and higher order derivatives of the variables [11].

Locally, for the case of a fixed-wing UAV, \mathcal{C} may include position in three-dimensional space, as well as yaw angle (heading). The roll and pitch angles and rates, as well as the vehicle velocity must then be represented in \mathcal{Y} .

Although the state space representation of dynamics is very popular, it is not feasible for use in the motion-planning problem. The reason for this is that, as the number of states increases, the volume of the region in which the motion-planning algorithm must search increases exponentially, making the problem computationally intractable for non-trivial systems [11]. Due to this phenomenon, generally referred to as the *curse of dimensionality* [12; 13], a less complex means by which to encapsulate the dynamics of a vehicle is required.

2.7.2 Encapsulating System Dynamics Using Manoeuvres

An alternative method of capturing vehicle dynamics, is to use *manoeuvre space* representation. The term *manoeuvre* is usually applied, in the general sense, to any deliberate, controlled change in the motion of a vehicle. We shall, however, use the term in a more specific sense, as it is applied in several publications on automation [1; 11; 14], to mean a planned transition between two states of trim motion which encapsulates the dynamics of the vehicle. Trim motion, or trim flight, in the case of a UAV, is understood to refer to a condition where the vehicle is in a state of equilibrium relative to its body axes [11].

For a rotary-wing UAV, Frazzoli *et al.* [11] distinguish between trim states and transition manoeuvres. The former represent states in a manoeuvre state automaton and the latter, transitions between these states. Although we will occasionally refer to both a trim state and the transition manoeuvre from the preceding trim state collectively as “a manoeuvre” where no confusion can occur, one should bear in mind that these two distinct concepts are implied.

We specify a transition manoeuvre, and associated *coasting time* τ for the subsequent trim state as an ordered pair of manoeuvre identifier and coasting time for the subsequent trim state, (m, τ) where $m \in \mathcal{C}_M$, a finite collection of all manoeuvres.

The dynamics for each manoeuvre are encapsulated by a separate state space model. The continuous controllers for each manoeuvre may be completely separate and even accept different state variables as input [2]. The implementation of each manoeuvre is thus abstracted from the motion planner.

If we assume invariance of vehicle dynamics with regard to translation, which is reasonably accurate for a UAV if the altitude range – and therefore air density range – is

relatively narrow, the complete state of the vehicle after executing a manoeuvre can be determined knowing the initial state and the specific manoeuvre. More formally, this means that we may define the *manoeuvre space* as [11]

$$\mathcal{M} = \mathcal{C} \times \mathcal{C}_{\mathcal{M}}. \quad (2.7.5)$$

The benefits of using the manoeuvre space representation approach, as opposed to state space representation, are summarised by Van Daalen [2]:

1. Significantly fewer variables are required as an input to the system.
2. Encapsulation of the vehicle dynamics by manoeuvres eliminates the need to explicitly ensure conformance to the dynamic restrictions of the vehicle.
3. The mean and covariance of the vehicle state may be propagated once only, off-line for every manoeuvre, as opposed to performing the calculation on-line.

The aforementioned factors lead to a great reduction in computational complexity for the motion planner. Unfortunately, restricting the motion planner – and therefore also the vehicle using the motion planner – to a finite set of manoeuvres, leads to a reduction in overall manoeuvrability. If the manoeuvre set is chosen carefully such that it is representative of the most common real-world pilot manoeuvres for a vehicle, this effect may be minimal. Furthermore, a reduction in manoeuvrability does not necessarily imply any reduction in reachability.

The relevant topics are discussed in greater detail in Chapter 4, specifically as they relate to the case of a fixed-wing UAV.

Chapter 3

A Kinodynamic Planning Algorithm for Dynamic Environments

3.1 Overview

The motion planning problem for a vehicle, such as a fixed-wing UAV, is a specialised case of the *generalised movers' problem*: the problem of planning a conflict-free path for a robot through an environment which contains obstacles [15].

In this problem, we must consider two types of constraints. These are [16]

1. local constraints, representing the dynamics of an underactuated vehicle, and
2. global constraints, originating from the environment which the vehicle must navigate.

We therefore require a motion planning algorithm which, to iterate the requirements set in Section 1.3

1. guarantees *completeness*, i.e. that a solution trajectory will be found provided that at least one solution exists in a given environment,
2. generates solution trajectories which conform to both local and global constraints,
3. ensures that generated trajectories are free of conflict, and
4. is able to plan trajectories in real-time in order to provide alternative solutions should previously planned trajectories become conflicted.

3.2 Selecting an Algorithm for Motion Planning in Dynamic Environments

Bearing the aforementioned requirements for the planning algorithm in mind, we proceed with a brief literature study of existing motion planning algorithms. We begin by eliminating an entire class of motion planning algorithms which are obviously inappropriate for our problem, and proceed with a more detailed discussion of the methods which are considered to be suitable.

3.2.1 Combinatorial Algorithms

The first category of motion planning problems that we consider are *combinatorial* algorithms. These algorithms are complete and exact, meaning that they are guaranteed to find a solution to any planning problem, provided that at least one solution exists, within bounded time.

The major drawback of combinatorial algorithms is that their execution times increase exponentially or even double-exponentially with the number of dimensions in which they search [15]. These dimensions correspond to the states necessary to represent the vehicle position and orientation relative to the inertial reference axes: the degrees of freedom for the given vehicle [15]. For a fixed-wing aircraft, these are at least three position coordinates and three rotational angles, as well as the derivatives of some of these parameters required in order to encapsulate certain non-holonomic constraints. For such a large number of parameters, combinatorial algorithms become computationally intractable [16].

As a result, there is a general consensus that combinatorial algorithms are not appropriate for real-time motion planning applications, and such algorithms have been widely abandoned in favour of sampling based algorithms [16; 14; 11].

3.2.2 Sampling-based Algorithms

An alternative to the complete and exact combinatorial algorithms, is a class of sampling-based algorithms, which sample the vehicle input space or manoeuvre space, the latter being defined in Section 2.7.2.

A common approach in path-planning for holonomic robots is a sampling approach based on a regularly-spaced meshed-grid of samples. Construction of such a grid in the high number of dimensions required to describe the dynamics of a non-holonomic vehicle is extremely expensive, and the problem quickly becomes intractable as the grid spacing is reduced. An alternative approach, which is elegantly simple, is to employ randomised uniform sampling in the search space. This has been shown to lead to algorithms which are *probabilistically complete* – that is, the probability of a solution being found approaches one – in the limit as the number of samples taken approaches infinity.

A third approach, which has been shown to yield superior performance to randomised sampling for a relatively small number of dimensions (about ten or fewer) [17], is advanced deterministic sampling, based, for example, on low-discrepancy lattices using Hammersley or Halton sequences [17]. This approach retains the completeness property of the various algorithms to which it is applicable [17].

3.2.2.1 Incremental Search Algorithms

Incremental search algorithms expand a graph of states which are known to be reachable by sampling the vehicle input/manoeuvre space. A sampled constant command is applied to the vehicle and the result of this input is simulated for a fixed time interval. The trajectory induced by this input is then checked for conflict. If the probability of conflict is acceptably low along the trajectory, the final state of the vehicle after executing the proposed input is added to a graph structure, with the corresponding input representing the edge connecting the previous (parent) state to the new (child) state.

For incremental search algorithms, the goal is usually represented by a region and not a discrete state. The incremental algorithm only terminates when a reachable state which happens to reside in this goal region is reached. This means that the class of incremental search algorithms is not specifically goal-directed, but rather relies on fast and complete exploration of the environment in order to reach the goal region. This unfortunately leads to situations in which the incremental search algorithms seem undirected or aimless in large, uncluttered environments which may be trivial for roadmap algorithms, which are presented in Section 3.2.2.2.

A notable early example of an incremental search algorithm applicable to vehicles with non-holonomic constraints, is an extended version of the Randomised Path Planner (RPP) [18; 19] by Barraquand and Latombe. This algorithm first constructs an artificial “potential field”, assigning attractive potential to the goal region and repulsive potential to obstacles in the environment [20]. Subsequently, a path is planned which descends along the steepest potential gradient. When stuck in a local minimum, a random motion is used to escape [20]. This process is repeated until the the global minimum in the potential field (i.e. the goal region) is reached.

A major drawback of this method is that it is incompatible with a probabilistic description of the external environment, as it relies on an explicit description of objects in the environment in order to construct a sensible potential field representation [2].

A method more suitable to an uncertain environment is that of Hsu *et al.* [21; 14], which has been proven in dynamic, uncertain environments for a vehicle with significant non-holonomic constraints [14]. This algorithm uses the same general philosophy as more basic incremental algorithms in order to extend a graph of known reachable states, but improves rapid diffusion of the graph through a weighted-randomised approach in selecting which node of the graph to extend. The weighting factor for each node is inversely proportional to the density of nodes in a region around it [21; 14].

The method of Hsu *et al.* is known to be probabilistically complete and it has been proven that the probability of this algorithm finding a solution in an *expansive* space¹ converges exponentially with the number of samples taken.

3.2.2.2 Roadmap Algorithms

Another class of sampling-based algorithms, generally termed *roadmap algorithms*, are distinguished from incremental algorithms mainly in that they sample the vehicle state space, or a subset thereof, as opposed to the input or manoeuvre space. In general, roadmap algorithms construct a directed graph, rooted at the initial vehicle state and ideally terminating at the goal, through intermediate sampled *milestone* states. Milestone states are connected to the existing graph of reachable states using an algorithm referred to as a *Local Planning Method* (LPM). Unlike incremental search algorithms which plan trajectories to within an acceptable goal region around each milestone, roadmap methods connect milestone states exactly by applying the LPM.

We disregard the Ariadne’s Clew roadmap algorithm, which makes use of deterministic sampling, as we know that it is not applicable to non-holonomic vehicles [22].

¹Informally, a free space, \mathcal{F} is said to be expansive if every large sub-region of that space, $\mathcal{R} \subset \mathcal{F}$, has a large *lookout*. That is to say, every \mathcal{R}_m has a large region from which it is trivial to plan a trajectory into another sub-region \mathcal{R}_n . A more formal definition of expansiveness is given by Hsu [14].

The Method of Kavraki et al. Other roadmap methods which employ randomised sampling, are referred to as a *probabilistic roadmap methods* (PRMs), formally introduced by Kavraki et al. [23]. In the original definition, the PRM consists of two distinct phases.

The first phase is the *graph construction phase*, executed off-line, in which milestone states are sampled and a connection to the graph is attempted using the LPM [20]. The node in the graph to which each connection attempt is made, is selected using some distance function, the definition of which varies between implementations [20].

The second, on-line, phase of the PRM is the *query phase*, in which the algorithm searches the graph built during the first phase for a feasible, conflict-free path between the initial state and the ultimate goal state [20].

Extended versions of the PRM, which are applicable to dynamic environments, exist. These extended PRMs search in the state-time space and integrate the two distinct phases of the original PRM into a single on-line, iterative process [2]. So-called *lazy conflict checking* is used in this case, wherein conflict is only checked on specific candidate trajectories which may be used, instead of throughout the entire PRM graph [2].

The Rapidly-exploring Random Tree (RRT) Many modern roadmap algorithms are based on the Rapidly-exploring Random Tree (RRT) [24; 16; 22]. The RRT samples milestones and chooses a constant vehicle manoeuvre or input which brings the vehicle as close as possible to the milestone within a predefined time interval [24; 22]. The newly proposed trajectory is then checked for conflict and, if required, trimmed short, based on predicted conflict along the path. A new milestone, corresponding to the terminal state in this trajectory, is then added to a tree data structure of known reachable states. As its name implies, the RRT is designed to explore available free space as quickly as possible. One obvious drawback is that the RRT is not able to reach a milestone exactly. If we wish to circumvent this shortcoming in reaching the final goal state, we may employ an LPM which is able to connect this final state directly.

Frazzoli Probabilistic Roadmap (PRM) Alternatively, an algorithm proposed by Frazzoli [1; 11] addresses this shortcoming of the RRT by attempting to connect sampled milestones *exactly* to each node in a graph of known reachable states using an LPM. The order in which this connection is attempted is determined by a sorting criterion. In Frazzoli's implementation, the milestones already in the graph are sorted according to the cumulative cost from the initial state to each milestone [1; 11], although different sorting criteria may be used. If a connection is successfully completed by the LPM, the new milestone is added to the graph, along with an additional milestone, chosen at a random point on the trajectory connecting the new milestone to the graph [2; 11].

In each iteration, Frazzoli's PRM also attempts a connection between the graph of connected milestones and the final goal state. If such a connection is successfully completed using the LPM, the graph is pruned: all branches which cannot possibly yield a solution with lower cost than the current best solution are discarded [1; 11]. In summary, Frazzoli's PRM iterates through the four steps:

1. Sample milestones.
2. Attempt to connect new milestones to a graph of known reachable states using the LPM, checking for conflict.

3. Attempt to connect the graph to the goal state.
4. Prune the graph of excess branches with excessive cost when a solution has been found.

Variations and Extensions of the PRM and RRT Extensions to the standard PRM and RRT, called PRM* and RRT* respectively, were introduced by Karaman and Frazzoli [25]. The major proposed modification to the original algorithms, is that connection to milestones be made exclusively within spheres of radius $\log n/n$, where n is the total number of sampled milestones [25]. The PRM* and RRT* algorithms are known to be asymptotically optimal, unlike their unmodified counterparts, meaning that the cost of trajectories generated by these algorithms is guaranteed to converge asymptotically to the minimum possible solution [25]. Due to the relatively minor impact of this modification to the original structure of the algorithms, we consider the RRT* and PRM* minor variations on the standard RRT and standard PRM, respectively, in terms of implementation effort.

Various additional improvements to the PRM are proposed by Van Daalen [2]. Firstly, in order to address uncertain environments, the conflict detection problem is separated from the planning algorithm through the use of an external conflict detection module. This allows the planning algorithm to take uncertainty of the environment and vehicle states into account through a probabilistic implementation of the conflict detection module, without affecting the planning algorithm itself, other than requiring the storage of cumulative probability of conflict along each trajectory as opposed to a binary *conflicted* or *not conflicted* value.

Secondly, the PRM is extended to sample in time, in addition to the subset of vehicle states, in order to be applicable to dynamic environments [2].

Furthermore, the process of generating a secondary milestone when connecting a new milestone to the graph is abandoned, since it is shown to lead to unwanted clustering of milestones [2].

Finally, and most importantly with regard to performance, a technique is introduced to bound the sampling region dynamically [2]. This topic is explained in greater detail in Section 3.3.2.

Based on the overview of motion planning algorithms, the candidates which seem promising for application to a dynamic, cluttered environment are the RRT and the PRM, especially using the extensions proposed by Van Daalen [2].

Neither option is clearly superior to the other in terms of performance, other than for very specific cases. We choose the PRM/PRM*² due to its ability to connect milestone states directly using a local planning method, and proven reliability in dynamic, cluttered environments [11; 2].

3.3 Motion Planning Algorithm Implementation

We now consider the PRM motion planning algorithm [1; 11] with many of the enhancements proposed by Van Daalen [2]. Due to the additional complexity required and the

²At the time of implementation, the PRM* and RRT* [25] algorithms were not yet published. Our implementation is therefore based on the original PRM, but our approach in developing a generic local planning method (LPM) is equally applicable to either variation.

fact that the aim of the project lies within the local planning problem and not the larger motion planner, we do not initially add the extensions provided by the PRM* [25]. The relevant enhancements for the PRM* algorithm may be added at a later stage without rendering any significant part of the original PRM redundant or incompatible.

3.3.1 The Probabilistic Roadmap Method Algorithm

Pseudocode for the PRM algorithm, based on the work of Frazzoli [11] and Van Daalen [2], is presented as Algorithm 1.

The PRM algorithm initially attempts a direct connection between the initial and goal state-time pairs using the LPM (line 2 in Algorithm 1). The LPM is discussed at length in Section 4.3. For now, it may be considered a black-box function which is known to return either the optimal input connecting two state-time pairs or a result indicating failure.

If the LPM successfully connects the initial and goal state-time pairs, and the trajectory which represents this solution is found to be unconflicted (line 3 in Algorithm 1), the PRM terminates, as this solution is known to be optimal. If the LPM fails, the main iterative loop of the PRM begins. The PRM iteratively samples milestone states, attempting to connect these to a graph data structure, rooted at the initial vehicle state-time pair. The LPM is then applied to find a solution by attempting a connection between milestone nodes in the graph and the goal state-time pair.

Algorithm 1 Pseudocode outlining the Probabilistic Roadmap Method (PRM) Algorithm

```

1: function PRM( $\mathbf{x}_0, t_0, \mathbf{x}_g, t_g$ )
2:    $\mathbf{u} \leftarrow \text{LPM}((\mathbf{x}_0, t_0), (\mathbf{x}_g, t_g))$ 
3:   if ConflictDetect( $\mathbf{u}$ )  $< \xi$  then           ▷  $\xi$  a constant denoting max. conflict threshold
4:     return  $\mathbf{u}$ 
5:   else
6:     Initialise graph  $\mathcal{G}$ 
7:     while not terminated do           ▷ Some cutoff time may be imposed, if necessary
8:        $\mathcal{X} \leftarrow \text{SampleMilestones}()$            ▷ Refer to Section 3.3.2
9:        $\mathcal{X} \leftarrow \text{Sort}(\mathcal{X})$ 
10:      for each  $(\mathbf{x}_s, t_s)$  in  $\mathcal{X}$  do
11:        Connect( $(\mathbf{x}_s, t_s), \mathcal{G}$ )           ▷ Refer to Section 3.3.3
12:      end for
13:      if Connect( $(\mathbf{x}_g, t_g), \mathcal{G}$ ) then
14:        Prune( $\mathcal{G}$ )           ▷ Refer to Section 3.3.4
15:      end if
16:    end while
17:    return current solution from  $\mathcal{G}$ 
18:  end if
19: end function

```

Every time a new solution is found which is of lower cost than the current best solution, the graph is trimmed of excess branches of higher cost.

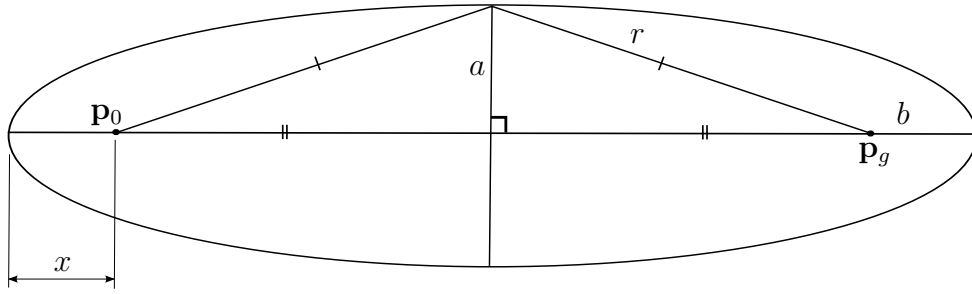


Figure 3.1 – Construction for determining the spatial axis lengths of a sampling space which incorporates a maximum velocity constraint.

3.3.2 Sampling from a Bounded Subset of the State-Time Space

In order to be able to address dynamic environments, we wish to generate milestone samples in a subspace of the vehicle state space (for example, position and heading), with associated time components. We shall refer to the combined state-time space from which we wish to sample as the *sampling space* \mathcal{S} . Note that \mathcal{S} does not necessarily span the entirety of the dimensions in the state or time space. For example, due to some constraint on the vehicle, such as its maximum velocity, some parts of the space may be inadmissible. The discrepancy between the size of \mathcal{S} and the entire state-time space is notable in very large environments. If we do not bound the sampling space appropriately, our motion planner becomes dependent on the size of the environment and may struggle to find solutions in environments which are very large, even when these environments contain little clutter.

Ideally, we wish to restrict the sampling space to the smallest possible size which does not prevent the motion planning algorithm from finding a solution. One proven method of reducing the size of the sampling space, is by enforcing rejection-sampling based on a subset of the dynamic constraints on the vehicle for which we can devise simple geometric tests [2]. For example, given the initial time and a constraint on the maximum time at the goal state, we can define a simple geometric test to reject states with associated positions which are unreachable when also taking the maximum velocity of the vehicle into account [2].

We use a novel approach to incorporate simple constraints in the definition of sampling space itself. Consider the maximum velocity constraint mentioned previously. Incorporating only this constraint in \mathcal{S} , the projection of \mathcal{S} in the three spatial dimensions is given by an ellipsoid, with the length of the axes determined by the vehicle's maximum velocity restriction along the axes. The orientation of the ellipsoid is defined by a unit vector parallel to the directed vector from initial position, \mathbf{p}_0 , to final position, \mathbf{p}_g . The foci of the ellipsoid are at \mathbf{p}_0 and \mathbf{p}_g , the initial and goal state-time pairs.

In order to derive the axis lengths for the ellipsoid, consider a two-dimensional ellipse with axes coinciding with two axes of the three dimensional ellipsoid (i.e. a section of the ellipsoid along a plane which contains two of its axes). Let a be the semiminor axis of the ellipse and b , the semimajor axis. The distance from \mathbf{p}_0 to the closest major-axis vertex or, equivalently, from \mathbf{p}_g to the corresponding vertex on the opposite side of the major axis is x . This ellipse is presented in Figure 3.1.

If the vehicle travels only along the major axis at maximum velocity v_{max} , the furthest path it may follow in the total time, $\tau_{max} = t_g - t_0$, is to first travel the maximum distance

in the direction opposite to the goal (to the vertex furthest from t_g) and then to the goal state; that is to say

$$s_{max} = 2x + (b - x) + (b - x) = 2b, \quad (3.3.1)$$

which means that the semimajor axis

$$b = \frac{v_{max}(t_g - t_0)}{2}. \quad (3.3.2)$$

Now construct a line connecting \mathbf{p}_0 and either vertex on the semiminor axis as well as a line connecting \mathbf{p}_g to the same vertex. The result of the construction is two congruent right-angle triangles with common side (the semiminor axis), and, additionally, equal sides of length $(b - x)$, as is clear from Figure 3.1. Let the hypotenuses of the triangles be r . For the vertices on the semiminor axis to fall just inside the valid sampling region, it is necessary that $r = b$. Applying the Pythagorean theorem to either triangle, we find that

$$a = \sqrt{b^2 - \frac{|\mathbf{p}_g - \mathbf{p}_0|^2}{4}}. \quad (3.3.3)$$

According to the original definition of the two-dimensional ellipse axes, the values $2a$ and $2b$ correspond to the lengths of two of the axes of the three dimensional spatial projection of the sampling region. The derivation for the third spatial axis is similar to the derivation for the semiminor axis length of the two-dimensional ellipse.

Now that the spatial bounds of the ellipsoid are known, we may generate uniformly distributed positions therein (i.e. in the spatial dimensions of \mathcal{S}).

First we generate a uniformly distributed point, $\mathbf{p}_r = [x_r \ y_r \ z_r]^T$, in a unit sphere. The point may be found using parametric equations for a sphere but, in order to avoid a bias towards its origin, we need to generate points using the square root of the radius, rather than the radius itself [26]. We then have:

$$\mathbf{p}_r = \begin{bmatrix} \sqrt{r}\sin(v)\cos(\chi) \\ \sqrt{r}\cos(v)\cos(\chi) \\ \sqrt{r}\sin(\chi) \end{bmatrix}, \quad (3.3.4)$$

with

$$\begin{aligned} r &\sim U(0, 1), \\ v &\sim U(-\pi, \pi), \text{ and} \\ \chi &\sim U\left(-\frac{\pi}{2}, \frac{\pi}{2}\right). \end{aligned}$$

Next, a transformation is applied to deform \mathbf{p}_r to fit a uniform distribution over the volume of an ellipsoid with the appropriate axis lengths. This deformation matrix, \mathbf{T}_d , is simply a 3×3 diagonal matrix with the ellipsoid semi-axis lengths on the diagonal.

The resulting point is subjected to the transformation matrix \mathbf{T}_r , which rotates the point to coincide with the orientation of the axes of \mathcal{S} . Finally, the point is translated with the origin offset of the sampling space, \mathbf{t}_t .

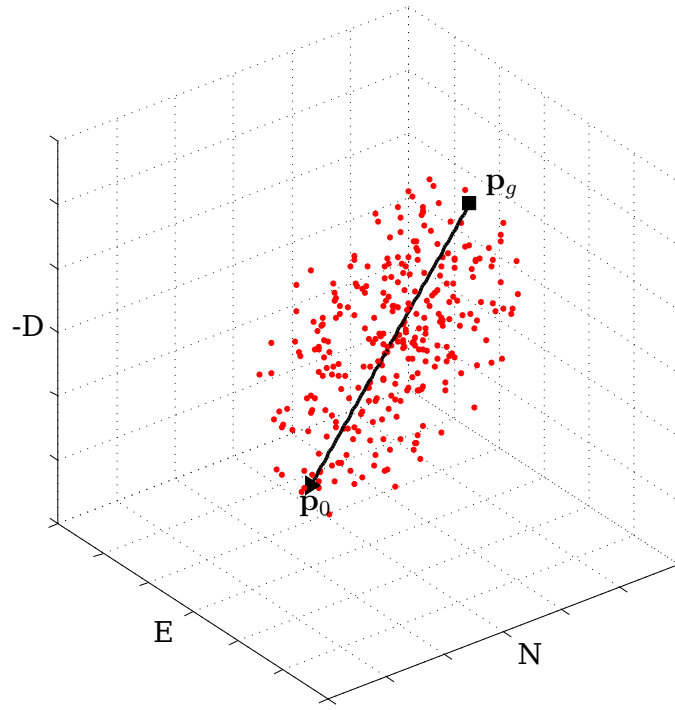


Figure 3.2 – Example of a sampling space which incorporates a maximum velocity constraint. The projection of the sampling space in the spatial dimensions is shown.

The result of this transformation is a point, \mathbf{p}'_r , uniformly distributed in the spatial dimensions of \mathcal{S} :

$$\mathbf{p}'_r = \mathbf{T}_r \mathbf{p}_r \mathbf{T}_d + \mathbf{t}_t. \quad (3.3.6)$$

The maximum and minimum times which may be associated with \mathbf{p}'_r are

$$t_{min} = \frac{|\mathbf{p}'_r - \mathbf{p}_0|}{v_{max}} \text{ and} \quad (3.3.7)$$

$$t_{max} = t_g - \frac{|\mathbf{p}_g - \mathbf{p}'_r|}{v_{max}} \quad (3.3.8)$$

respectively. Therefore, to generate the associated time, we take the sample time as:

$$t_r \sim U(t_{min}, t_{max}). \quad (3.3.9)$$

We are now able to generate a sample, $\mathbf{s}_r = (\mathbf{p}_r, t_r)$, from a uniform distribution over a space in which all samples conform to the maximum velocity constraint of the vehicle.

This efficient approach eliminates a large number of invalid state-time pairs. Additional vehicle constraints for which we can devise efficient tests may be incorporated into the sampling region using a similar approach. In practice, however, the sampling space will still always be a superset of the actual admissible milestone states, unless we are able to incorporate all the vehicle constraints in the definition of the sampling space. This is not practically feasible, as some non-holonomic constraints are not trivial to express using simple geometry.

In order to reduce the number of invalid samples, rejection sampling, based, for example, on the minimum possible path-length cost of a trajectory from the initial state to the goal state via a milestone state, may also be applied [2]. Once at least one solution has been found by the motion planning algorithm, we can further reject milestones based on the minimum cost J_{\min} of the current best solution trajectory.

In the case that no specific goal time is associated with the goal state, we may adapt this approach to initially automatically choose a time at goal, based on the maximum velocity constraint of the vehicle. While the planning algorithm fails to find a solution, the goal time may gradually be increased, in order to slowly inflate the sampling region until a solution is found [2]. At this point, the goal time which leads to the initial solution may be used as an absolute maximum to bound the size of the sampling region, after which the entire process may be repeated in order to find lower cost solutions, in terms of time to goal.

Milestones are usually sampled in a batch, and are often sorted in order of cost from the initial state-time pair to ensure shorter distances to new milestones. This is beneficial, as shorter trajectories are less likely to be obstructed, leading to a rapid increase in the number of connections within the PRM graph [2].

3.3.3 Extending the Graph of Reachable Milestones

Once a milestone state has been sampled and has passed all efficient rejection tests, we require a means by which to connect it optimally to a known reachable state. In this case, optimality may be defined in different ways, depending on the requirements for the motion planner. In the simplest case, it may simply be related to shortest path length. A more complicated cost function, involving other parameters, for example energy expended, is also possible. The algorithm responsible for this optimal connection is the LPM.

In order to determine to which milestone node in the graph a new milestone will be connected, the nodes in the graph are first sorted. As in Frazzoli's [11] implementation of the PRM, we sort according to cost between the new milestone and nodes already in the graph when no solution exists. This expedites connections and promotes fast exploration. Once a solution has been found, nodes are sorted by cumulative cost from the root node, which promotes low cost solutions. An alternative initial sorting based on the Euclidean distance between new milestones and the goal state is also found to work well for many environments.

The LPM is applied to each node in the graph in conjunction with the new milestone, until a valid connection is made. This connection is tested for conflict. In practice, the conflict detection is performed by an external conflict detection module. For purposes of the simulation, the conflict detection module is emulated by a mock conflict detection function, which is detailed in Appendix A.

If a trajectory generated by the LPM is determined to be unconflicted – or if the cumulative probability for conflict up to the new milestone is acceptably low for a probabilistic implementation of conflict detection – the new milestone is added to the graph of known reachable milestones. The edge which connects the new milestone to the graph represents the input resulting in the transition of the vehicle between the parent milestone node and the new milestone which becomes its child. The resulting graph structure is illustrated in Figure 3.3.

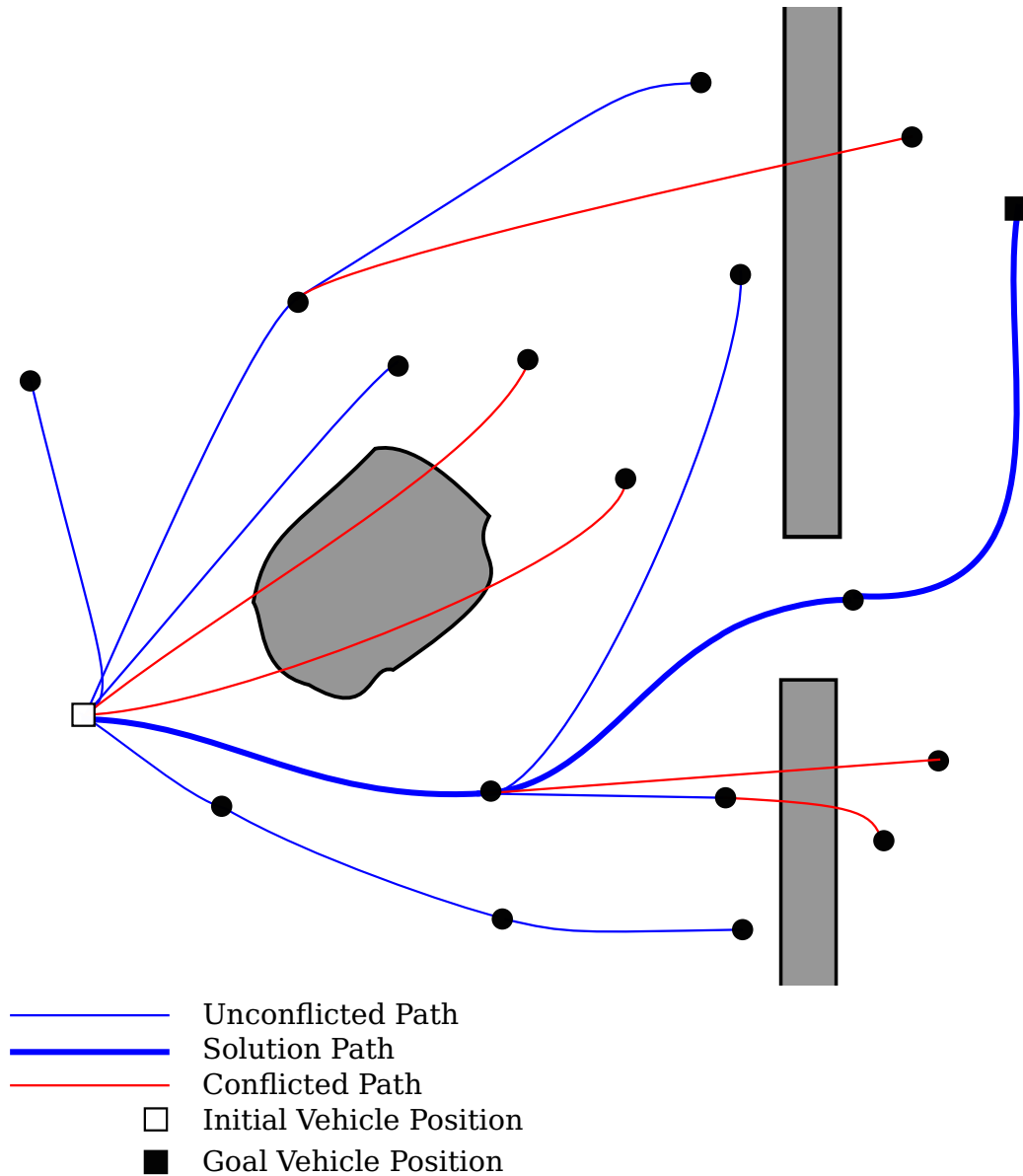


Figure 3.3 – Typical PRM graph in a cluttered environment.

3.3.4 Finding Solutions and Pruning the Graph

In each iteration of the PRM, every node in the graph is subjected to a connection attempt with the goal state-time pair. If such a connection is made successfully by the LPM, we have found a solution, represented by the sequence of edges in the graph which join the initial state-time pair to the goal state-time pair.

Every time a solution is found, the cost of the minimum solution, J_{\min} as defined in Section 3.3.2, is lowered if the new solution is of lower cost. If not, the cost is left unaltered. Next, the graph is trimmed of all branches which cannot possibly yield a lower cost solution than J_{\min} as verified using the efficient geometric tests of Section 3.3.2.

The sampling space is also adjusted, as explained in Section 3.3.2. The main loop of the PRM then repeats.

3.3.5 A Note on Implementation of the Motion Planning Algorithm

The outer loop of the PRM-algorithm is implemented in Matlab script, which lends itself to fast prototyping. The graph data structure is, however, implemented in ISO/IEC 9899:1999 specification C due to a lack of appropriate data structures in Matlab and its limitations with regard to memory management. As the PRM itself is not a main focus of this project, and we are more interested in the performance of the LPM, we are content with the relatively slow execution speed of loop code structures in Matlab script, even though it is likely to have an adverse affect on execution times of the iterative PRM-algorithm. It is, however, important that we are able to assess the absolute performance of the LPM. For the previously stated reasons, Matlab script is not considered appropriate for the LPM, and we therefore prefer to implement this algorithm in pure C. This code is compiled into a DLL which is accessible from within the PRM Matlab script.

For details on the implementation of conflict detection, refer to Appendix A.

Chapter 4

Local Planning In a Cluttered, Dynamic, Three-Dimensional Environment

4.1 Overview

As discussed in the previous Chapter, an efficient method to connect two states directly and optimally is required by the larger motion planning algorithm. A function responsible for performing this optimal connection is generally referred to as a Local Planning Method (LPM).

Since a fixed-wing aircraft is a non-holonomic vehicle; that is, a vehicle which is subject to certain velocity constraints which are not derivable from position constraints, classical motion planning approaches which search the vehicle state space directly are not applicable to our vehicle, and we therefore prefer algorithms which search the vehicle's manoeuvre space, as was explained in Section 2.7.

This chapter proposes the definition of a finite set of motion primitives and the implementation details of an LPM which utilises these motion primitives or *manoeuvres*.

A common approach to defining manoeuvres for a vehicle with approximately constant velocity and bounded turning radius, is reviewed in Sections 4.2.1 to 4.2.3. This approach is based on a set of geometrically perfect primitives which are combined into sequences called Dubins curves [27].

In Section 4.2.4, it is argued that Dubins curves are an inefficient approximation to the natural manoeuvres of a fixed-wing aircraft, and an alternative method of defining manoeuvres is subsequently introduced in Section 4.2.5 to address the shortcomings of Dubins curves. This method uses implicit definitions derived from a simple control system for the vehicle which does not regulate axial accelerations of the vehicle aggressively, as opposed to the control approaches typically employed to follow Dubins Curve trajectories.

A novel, generic LPM, based on the Fletcher-Levenberg-Marquardt Algorithm (FLMA), which solves the local planning problem using implicitly-defined motion primitives, is presented in Section 4.3.

4.2 Developing Manoeuvres

First, let us consider the most commonly used set of ideal motion primitives, which result in trajectories which can be defined exclusively by sequences of line segments and bounded circular curves with fixed radius equal to the minimum turning radius for the vehicle under consideration. The two distinct primitive types (circle segment or line segment) are referred to as the vehicle *manoeuvres*; that is, constant actions which may be applied for a time duration τ . A continuous sequence of these motion primitives is called a *manoeuvre sequence*.

It stands to reason that a local path planner using motion primitives which conform to the dynamic constraints of a vehicle, with no discontinuities in state between these primitives, will generate only paths which are achievable for that vehicle. If we develop a motion planner which is able to integrate with an arbitrary manoeuvre set, we may adapt the planner to suit any vehicle through a substitution of the manoeuvre set with one which is known to encapsulate the dynamics of the particular vehicle. Furthermore, we may initially test such a path planner using simple geometric manoeuvre definitions and later substitute these with more realistic manoeuvre definitions for a particular vehicle. In this section, we first define a simple geometric set of manoeuvres based on Dubins curves, and continue to develop more realistic equivalent manoeuvres for a fixed-wing aircraft.

4.2.1 Dubins Curves in Two Spatial Dimensions

Dubins curves [27] are a set of motion primitive sequences, composed exclusively of line segments and circle segments, which is known to yield the distance-optimal path between any two points on a plane for a vehicle class known as the *Dubins car*. This type of vehicle conforms to the following properties [27]:

1. The vehicle maintains a constant velocity in a forward direction, and
2. has an absolute minimum turning radius, ρ_{min} .

Dubins showed that a maximum of three motion primitives (curves or lines) are required to construct the distance-optimal path between any two points in a plane [27] and, furthermore, that the collection of six manoeuvre sequences in Table 4.1 are sufficient in this regard [27; 22]. In every sequence in Table 4.1, each constituent motion primitive has an associated execution time, τ_n , which may be zero.

Table 4.1 – Necessary and sufficient Dubins curve manoeuvres in two spatial dimensions [27]. The symbol C denotes a circle segment of radius ρ_{min} and L denotes a straight line segment.

$C_{(left,\tau_1)}$	$C_{(right,\tau_2)}$	$C_{(left,\tau_3)}$
$C_{(right,\tau_1)}$	$C_{(left,\tau_2)}$	$C_{(right,\tau_3)}$
$C_{(left,\tau_1)}$	$L_{(\tau_2)}$	$C_{(left,\tau_3)}$
$C_{(left,\tau_1)}$	$L_{(\tau_2)}$	$C_{(right,\tau_3)}$
$C_{(right,\tau_1)}$	$L_{(\tau_2)}$	$C_{(left,\tau_3)}$
$C_{(right,\tau_1)}$	$L_{(\tau_2)}$	$C_{(right,\tau_3)}$

4.2.2 The Ideal Dubins Airplane

We assume that a fixed-wing aircraft adheres to Property 2 of the Dubins car, as stated in 4.2.1. Furthermore, such an aircraft must maintain at least a minimum forward velocity in order to generate sufficient lift to remain airborne. This velocity in a plane which is parallel to the surface of a flat Earth (the North and East or *horizontal* plane) may be approximated as being constant for a fixed-wing aircraft performing only non-acrobatic manoeuvres about trim condition, in which case the vehicle also adheres to Property 1 of the Dubins car. Dubins curves may therefore be applied to find the minimum-distance path between the projection of two points in three-dimensional space onto the two-dimensional horizontal plane.

Taking into account that the lateral and longitudinal modes of motion of a fixed-wing aircraft may, per common approximation, be decoupled, the vehicle's vertical path may now be considered independent of the horizontal motion. Determining the optimal vertical motion becomes a simple problem in one spatial dimension and time once the solution to the horizontal problem is known.

This effectively extends Dubins curves to three spatial dimensions, which may be used to determine minimum-distance paths for a three-space Dubins car, or Dubins airplane. The trajectories for these manoeuvres represent sequences of straight lines in three dimensions and helices about a vertical axis.

4.2.3 Defining Geometrically Perfect Three Dimensional Manoeuvre Sets

Manoeuvre sequences for a Dubins airplane are now defined as concurrent, yet independent horizontal and vertical component manoeuvre sequences, linked by common initial and final times, t_0 and t_f . The horizontal component of a manoeuvre set is defined by one of the six Dubins curve manoeuvre sequences in Table 4.1. For the sake of computational simplicity, the vertical manoeuvre sequence component is specified only as an equivalent constant vertical velocity, v_{vert} , across the entire duration of the manoeuvre sequence. It must, however, be noted that, due to dynamic velocity constraints, implementing the manoeuvre sequence thus is not plausible.

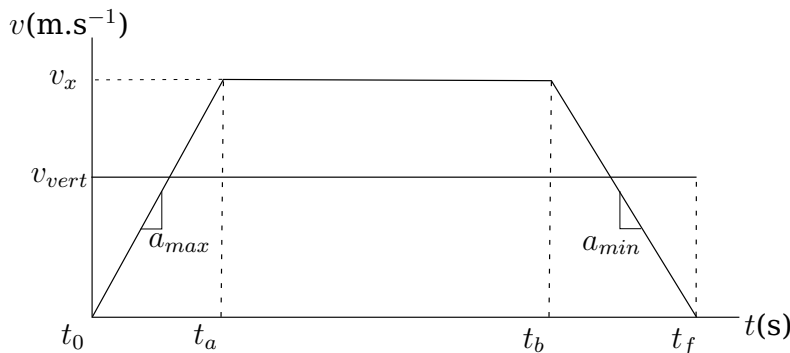


Figure 4.1 – Relationship between actual vertical velocity and equivalent constant vertical velocity for the Dubins airplane.

The actual vertical components of manoeuvre sequences are based on the assumption that approximately constant maximum upwards and downwards vertical accelerations,

a_{max} and a_{min} respectively, are possible. As seen in Figure 4.1, the vertical manoeuvre sequence component is defined as a constant vertical acceleration of either a_{max} or a_{min} ¹, up to a maximum vertical velocity, v_x (bounded by the maximum permissible vertical velocities), at time t_a . The opposite vertical maximum acceleration is then applied from time $t_b \geq t_a$ until the final manoeuvre sequence end time, $t_f = \sum_{n=1}^3 \tau_n$, resulting in a zero vertical velocity at t_f . The change in altitude is thus distributed across all three manoeuvres. The final vertical displacement resulting from the manoeuvre sequence must be equal to that resulting from the equivalent constant vertical velocity, v_{vert} , as defined above. The relationship between v_{vert} and actual vertical velocity is illustrated in Figure 4.1.

For $a = |a_{max}| = |a_{min}|$, it can be shown that the maximum actual vertical velocity

$$v_x = \frac{a}{2} \left(t_f - \sqrt{t_f^2 - \frac{4t_f v_{vert}}{a}} \right), \quad (4.2.1)$$

where

$$t_f = \sum_{n=1}^3 \tau_n, \text{ and} \\ v_{min} \leq v_x \leq v_{max},$$

with v_{max} and v_{min} the maximum permitted downward and upward (or minimum negative downward) vertical velocities for the aircraft. Then

$$t_a = \frac{v_x}{a}, \text{ and} \quad (4.2.3)$$

$$t_b = t_f - t_a = t_f - \frac{v_x}{a}. \quad (4.2.4)$$

Figure 4.2 shows an arbitrary manoeuvre sequence for the ideal Dubins airplane in three dimensions, as well as its projection in the horizontal and vertical plane. In this example, the horizontal component is composed of the $(C_{(left, \tau_1)}, L_{(\tau_2)}, C_{(right, \tau_3)})$ Dubins curve set and the vertical component is a steep climb. Note that the projection in the vertical plane is not a straight line for the entire duration of the manoeuvre sequence, but that it, instead, curves gradually to become parallel to the horizontal plane at its end points.

4.2.4 Practical Implications of Using Geometrically Perfect Manoeuvre Sets

We know from the work of Dubins [27] that the set of ideal motion primitives defined in the preceding section are sufficient in terms of reachability and, furthermore, we have evidence from the research of Gaum [5] which suggests that these trajectories are attainable for a real aircraft. Using geometrically perfect primitives simplifies the planning problem, as it is easy to express such trajectories using explicit formulae.

We must, however, also consider the drawbacks of constraining a real aircraft to these manoeuvres in real-world applications. In this section we argue that geometrically perfect motion primitives consisting of lines and circle segments are not energy-optimal manoeuvres for a fixed-wing UAV.

¹Depending on the relative vertical position components of the initial and goal states

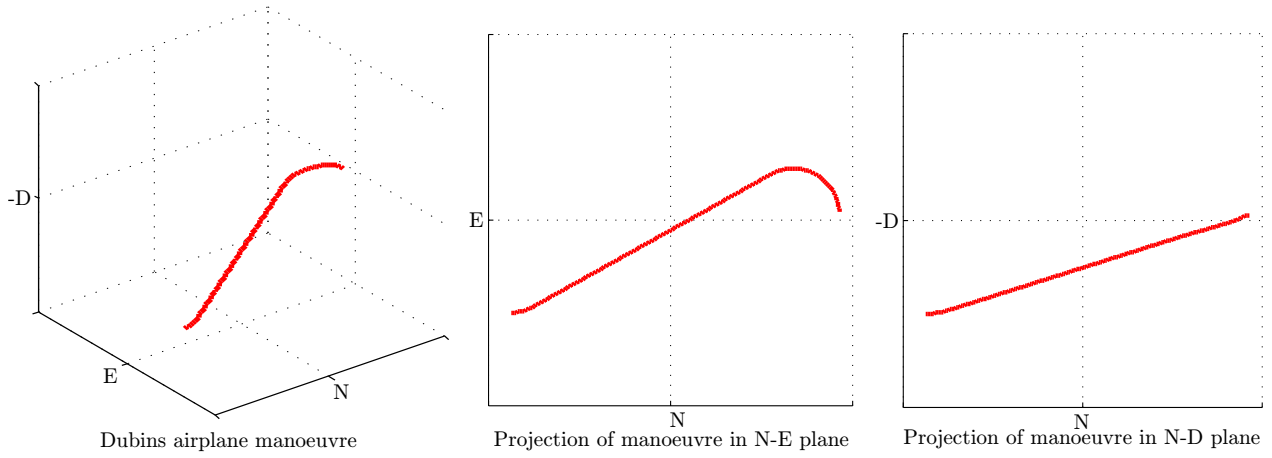


Figure 4.2 – Example of an ideal Dubins airplane manoeuvre sequence.

4.2.4.1 High Energy Consumption for Geometric Manoeuvres

In order to understand why geometrically perfect Dubins curves are inefficient in terms of energy consumption, we must consider the dynamics involved during typical real-world flight manoeuvres for a fixed-wing aircraft.

During straight and level flight, the thrust produced by the aircraft's engine must balance the aerodynamic drag on its body in order to maintain a constant velocity. This drag consists of two components: *parasitic drag* which is proportional to the square of the aircraft's velocity, and *induced drag*, which is inversely proportional to the square of the velocity. The induced drag is drag due to lift and is minimal for trim flight and significant for a large angle of attack [28].

Efficient real-world, non-acrobatic flight manoeuvres can be considered energy-optimal transitions between periods or states of trim flight. By energy-efficient, we mean that the manoeuvres tend to conserve the sum of the kinetic and potential energy, sometimes referred to as the *energy package* [29] of the aircraft, with minimal energy lost to drag. This becomes an important factor during the transition from straight and level flight into a turning manoeuvre.

When turning, in addition to overcoming drag, the aircraft must continuously generate a centripetal force to overcome its inertial tendency to continue in a straight line. This additional force is given by [28]:

$$F_c = \frac{mv^2}{r}, \quad (4.2.5)$$

where

- m is the inertial mass of the aircraft,
- v is the velocity of the aircraft, and
- r is the radius of the turning circle.

The centripetal force is typically generated by rolling or *banking* the aircraft at an angle ϕ , which lends a horizontal component to the lift generated on the wings. Intuitively, this leads to a decrease in the vertical component of the lift. In order to maintain constant altitude, additional lift must be generated through an increase in the angle of attack, which, in turn, leads to an increase in induced drag. The increased drag can only be

overcome by providing additional thrust. A minimum radius turn requires a relatively low speed and high banking angle [28]. The typical banking turn is illustrated in Figure 4.3.

Consider that the ideal Dubins manoeuvres require instantaneous transition between straight flight into a minimum radius turn at the transition from a straight line into a circle segment. This implies an instantaneous change in acceleration from zero to $\frac{v^2}{r}$. In other words, the aircraft must theoretically bank to a high angle instantly and achieve an infinite rate of change of acceleration or *jerk* if it is required to maintain airspeed. Intuitively, this discontinuity is extremely energy intensive to approximate and even a reasonable approximation must necessarily lead to some overshoot in trajectory and a deliberate dissipation of kinetic energy before entering the turn. This energy must subsequently be regained. This defies the goal of conserving the aircraft's energy package, and, strictly speaking, a fundamental requirement for a Dubins vehicle (i.e. constant forward velocity).

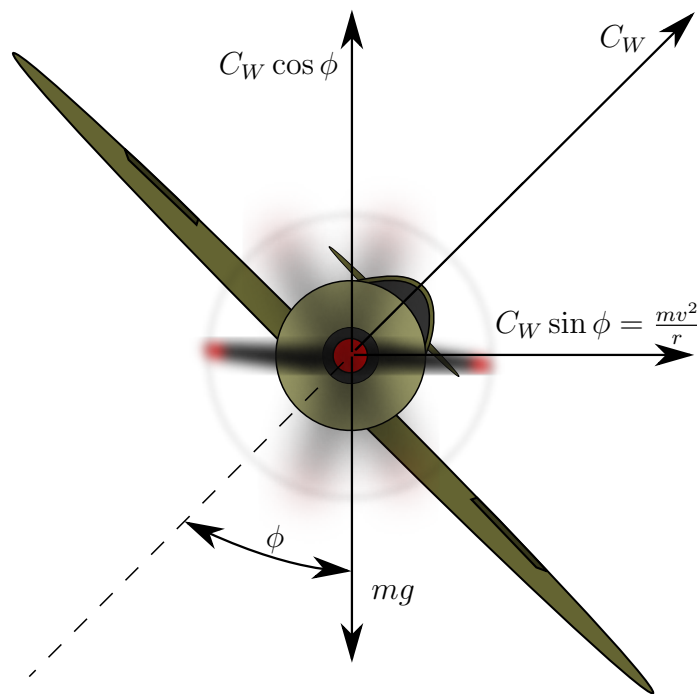


Figure 4.3 – Forces acting on a fixed-wing aircraft during a banking turn [28].

C_w is the total force generated by the aircraft along its z-body axis,
 m is inertial mass of the aircraft,
 v is forward velocity relative to the air, and
 r is instantaneous turning radius.

4.2.4.2 High Energy Consumption requirements of Path-Following

In order to trace trajectories such as the ideal set of Dubins curves while maintaining a constant velocity, we require a complex path-following control technique. Path-following for a fixed-wing UAV is a well-researched topic. The Specific Acceleration Matching (SAM) technique [7] summarised in Section 2.6.2 has been demonstrated to be an effective technique to facilitate path-following for a fixed-wing UAV [5].

This control approach has been applied successfully to a CAP232 aerobatic aircraft to track trajectories which represent Dubins curves in the horizontal plane [5]. The outer loop controllers of the system continuously calculate errors between actual position of the vehicle and a position-based reference trajectory. These errors are then translated into specific accelerations along the inertial reference axes, and a related roll rate [5].

The SAM control system therefore continuously minimises the vehicle's deviation from the planned position trajectory. If the vehicle is driven off its trajectory by an unbalanced external force, for example wind, a tracking error will occur and the control system will produce a specific acceleration counter to the direction of the error. This acceleration has a magnitude which is proportional to the severity of the tracking error. This implies that in windy conditions, the control system expends additional energy in counteracting the effect of the wind. This energy may be significant for a small UAV flying in moderate winds.

The ability to track a position-based trajectory accurately is vital if we require our UAV to traverse long distances, over extended periods, without an opportunity to re-plan its trajectory. We argue, however, that accurate position tracking is not necessarily beneficial in a dynamic environment with significant uncertainty in the future position of external objects. In such uncertain environments, it is constantly necessary to re-evaluate the probability of conflict with objects in the environment along the planned trajectory, and to re-plan when the probability of conflict becomes too high. This is determined by considering the position probability distribution of every object in the environment in conjunction with that of our vehicle. It is therefore a reasonable assumption that, for a sufficiently cluttered environment with high uncertainty in the future position of external elements, a low uncertainty in the future vehicle position does not significantly influence the overall uncertainty when predicting conflict. Since, as per our original problem definition, we assume no *a priori* knowledge of the nature or future trajectories of objects in the environment, we expect that the environments in which our vehicle will operate do indeed have high uncertainty with regard to the position of other objects.

We hypothesise that using a more "relaxed" control approach which does not make use of path-following techniques, will not significantly impact the uncertainty in predicting conflict with external objects, provided that we are able to re-run the kinodynamic planning algorithm periodically.

It is important to keep in mind that the uncertainty in the future configuration of the external environment is high regardless of the control approach used. This means that periodic re-planning is already a vital system requirement dictated by the nature of the environment and not a new requirement imposed by the use of a more basic control approach. The major benefit of using a simple control system which does not implement path following, is that it does not incur the unnecessary energy expenditure inherent to path following, and that it can be designed to implement manoeuvres without acceleration discontinuities and which are therefore more natural for a real-world vehicle with

non-holonomic dynamic constraints. The design for such a simple control system and a kinodynamic planner which is able to construct nominal trajectories using the motion primitives produced by the system are detailed in the following sections.

4.2.5 Practical Approximations to the Ideal Dubins Manoeuvres

Bearing in mind the assumption that our application does not require extremely accurate position-based tracking, we seek an alternative planning and control-approach which does not have excessive energy requirements. A straightforward approach, from a control systems perspective, is to implement a simple control system which is manoeuvre based, instead of trajectory based. We then require some means to capture the nominal trajectories which are produced by this simple control system and a planning method which is able to make use of trajectories which are defined implicitly by these manoeuvres.

4.2.5.1 A Simplified Control Approach

In order to achieve an approximation of the basic Dubins curve primitives, *straight line* and *circle segment*, which we shall henceforth term *banking turn* or *constant heading*, we require a system to control our vehicle to one of only three predefined motion primitive states: *left banking turn*, *right banking turn*, and *constant heading*. If velocity and altitude are regulated independently, each of the aforementioned primitives can simply be specified by a roll angle. This angle is zero for constant heading flight and some constant non-zero value $\pm\phi_c$ for a left/right banking turn. In practice, we also add a term to the z-body axis acceleration during banking turns to compensate for the loss in lift due to banking. Algorithm 2 outlines the mapping of a motion primitive M to appropriate values for the z-body axis acceleration A_z and roll rate $\dot{\phi}$.

Algorithm 2 Mapping Motion Primitives to Control Parameters

```

1: function SimpleControl( $M, z_g$ )           ▷  $M$  is manoeuvre identifier;  $z_g$  is goal altitude
2:    $A_Z \leftarrow \text{CalculateVerticalAcceleration}(z_g)$    ▷ Required z-body axis acceleration
3:    $\phi \leftarrow 0$ 
4:   switch  $M$  do
5:     case BANKING LEFT TURN
6:        $A_Z \leftarrow A_Z + a_{Z_{\text{bank}}}$            ▷ Compensate for loss in lift due to roll
7:        $\phi \leftarrow -\phi_c$                      ▷ Bank left to constant roll angle  $\phi_c$ 
8:     case BANKING RIGHT TURN
9:        $A_Z \leftarrow A_Z + a_{Z_{\text{bank}}}$            ▷ Compensate for loss in lift due to roll
10:       $\phi \leftarrow \phi_c$                      ▷ Bank right to constant roll angle  $\phi_c$ 
11:    $\dot{\phi} \leftarrow \text{CalculateRollRate}(\phi)$ 
12:   return  $A_Z, \dot{\phi}$ 
13: end function

```

As the inner loop controllers developed by Gaum [5] suit our needs and have been verified during several practical flight tests on two fixed-wing UAVs [5; 3], we leave them unaltered, and using aerodynamic coefficients for our test UAV as determined by [3]. We feed the values of A_z and $\dot{\phi}$, as determined using Algorithm 2, to the inner loop controllers. Note that Algorithm 2 and its helper functions are able to access the current

vehicle state. The SAM control is disabled through a built-in mechanism in the original control system and specific accelerations A_X and A_Y are set to zero. The modified control system is outlined in Figure 4.4.

Although this control approach is selected because it is expected to yield favourable characteristics with regard to energy consumption, it should be noted that we present it only as an illustrative example for the planning algorithm we develop, and not as a strategy for achieving optimal energy efficiency. Critically, and more relevant to the aim of our investigation, we proceed in a general manner in order to ensure that the final kinodynamic planning algorithm is generic and that it may be applied interchangeably to different control systems and vehicles.

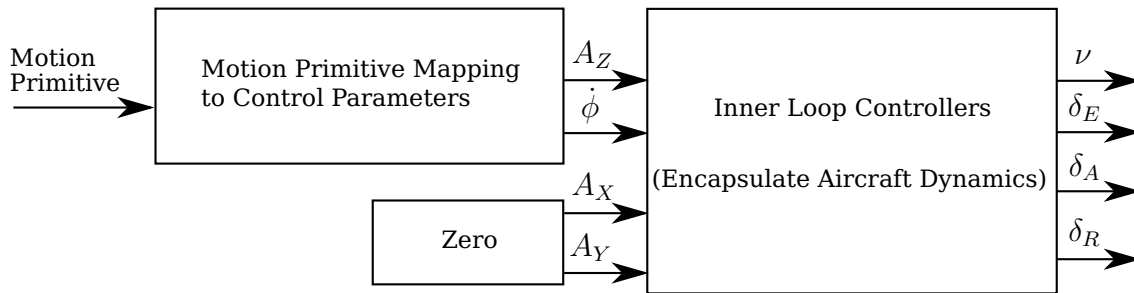


Figure 4.4 – Proposed Simple Control (SC) system architecture. Specifies constant vertical acceleration and roll rate based on the type of motion primitive required (banking turn or constant heading flight). Feedback paths are not shown, but may be inferred from Figure 2.3.

4.2.5.2 Capturing Nominal Trajectories for Practical Manoeuvres

For the simple control system proposed in the preceding section, motion primitives can no longer be described with reasonable accuracy using simple algebraic expressions. If we are to use this control system, we require manoeuvre definitions which correspond well to the manoeuvres which this control system is able to execute. We expect that it is possible to describe such manoeuvres using Bézier curves. This approach is, however, specific to manoeuvres generated using a particular control system for a specific vehicle. A more generic approach is desired.

To this end, we capture *nominal trajectories* empirically, by applying the control system to the simulated vehicle model under ideal conditions (i.e. no noise signals). Each motion primitive which cannot be described by a straight line (in the most basic case, left banking turn and right banking turn), is captured as a position-based trajectory at a high time resolution. Vertical motion is captured independently from horizontal motion. In practice, the control system will yield these nominal trajectories with random deviations due to wind and measurement noise. We may reduce this effect through replanning-techniques (investigated in Section 5.4.2).

The nominal trajectories generated are saved to secondary storage in a format which can easily be constructed into a hash-map structure in memory on first execution of the planning algorithm. The trajectories are stored as discrete position sequences.

Our planning algorithm, on the contrary, requires a continuous description of the nominal trajectory for each motion primitive. To achieve an appearance of continuity, we make use of second order interpolation between recorded data points².

The process of retrieving the nominal resultant position of the vehicle after executing a specific manoeuvre for a time t_{desired} begins by selecting the closest matching data point from the hash-map. The starting address for a contiguous block of memory A_{base} which represents the correct motion primitive is known. If the position on the trajectory at time t_{desired} is to be found, the address in memory of the closest stored data point is given by

$$A_{\text{index}} = \lfloor t_{\text{desired}}/T_s \rfloor D + A_{\text{base}}, \quad (4.2.6)$$

with

$$T_s \leq t_{\text{desired}} \leq NT_s,$$

and where

- T_s is the sampling time used in generating the stored trajectory data,
- D is the storage size of the *double* variable type in bytes,
- N is the total number of samples stored for the motion primitive, and
- A_{base} is the starting address of the block of memory which contains the desired motion primitive.

Next, we construct a quadratic curve which passes through the data point d_0 at A_{index} , representing time t_0 and the two adjacent data points, d_{-1} at t_{-1} and d_{+1} at t_{+1} .

In each spatial dimension, we wish to find the coefficients (c_2, c_1, c_0) in an equation of the form

$$c_2 t^2 + c_1 t + c_0 = d. \quad (4.2.7)$$

We can write this in matrix form as

$$\begin{bmatrix} t_{-1}^2 & t_{-1} & 1 \\ t_0^2 & t_0 & 1 \\ t_{+1}^2 & t_{+1} & 1 \end{bmatrix} \begin{bmatrix} c_2 \\ c_1 \\ c_0 \end{bmatrix} = \begin{bmatrix} d_{-1} \\ d_0 \\ d_{+1} \end{bmatrix}, \quad (4.2.8)$$

or, more compactly,

$$\mathbf{T}\mathbf{c} = \mathbf{d}. \quad (4.2.9)$$

In Equation 4.2.9, \mathbf{d} is a vector of relative positions along the North, East or Down axis. We determine an independent quadratic fit for the series of points along each axis. We solve for each vector \mathbf{c} using Gaussian elimination³.

This leads to three explicit equations, each representing an independent, continuous quadratic fit in a single dimension in the region close to the desired time. We substitute

²For reasons which will become apparent in Section 4.3.3, we require at least second order interpolation in order for the planning algorithm to function correctly.

³Our implementation of Gaussian elimination is an enhanced version of a function by Scvortov [30]

the actual value of t_{desired} into these quadratic equations to obtain a continuous three-dimensional position on the nominal trajectory of the motion primitive.

Although Gaussian elimination has a high time-complexity of $\mathcal{O}(n^3)$ for an arbitrary matrix, our Vandermonde matrix \mathbf{T} is square and only of size 3×3 . In practice, the order of magnitude of operations for Gaussian elimination can be reduced to $\frac{n^3}{3}$ if \mathbf{T} is known to be square [31]. The operation is therefore considered to be relatively inexpensive.

The position determined thus is relative to the origin of a convenient Earth-fixed system of axes and assuming an initial heading of zero radians. We make use of a translation and the DCM to displace and rotate the point in three-dimensional space to find the absolute position coordinate on the nominal motion primitive trajectory.

This entire process is encapsulated by a function, which takes as arguments the horizontal manoeuvre type or identifier (*banking turn* or *constant heading*), a corresponding execution time, a vertical component and current state vector for the vehicle. The function returns the nominal final state vector and time after executing the manoeuvre.

If we desire the entire position-based nominal path which is associated with a motion primitive (in order to perform conflict detection), we simply copy a contiguous block of memory starting at A_{base} for the desired motion primitive, and ending at the calculated A_{index} (Equation 4.2.6). We then rotate and translate the entire path as previously described. When calculating the trajectory of a subsequent manoeuvre, it is, however, important that the initial state be taken as that calculated using the continuous, interpolated state, and not the closest matching discrete state taken directly from memory. Erroneously using the discrete value introduces a small discontinuity between the end of one manoeuvre and the start of a subsequent manoeuvre. This effect may become significant over the course of a large number of consecutive manoeuvres.

4.3 Local Planning Method

Having defined practical manoeuvres for a UAV, a method is now needed to select a manoeuvre sequence and specify the execution times for its constituent parts in order to connect two distinct state-time pairs.

We need to specify only a subset of vehicle states which are relevant to the application of our vehicle. For example, if we are only interested in navigating waypoints, there is no reason to specify the heading angle of the vehicle at the goal state, but this parameter is vital if we require the vehicle to complete surveillance of a target area from a specific angle.

Not only does the reduction of the number of states in which we search reduce the time cost associated with the LPM-algorithm, but it avoids significant over-specification of the goal state. For example, it may be barely possible for a vehicle to reach a certain goal position at a specified time if the goal heading is unspecified, but for certain heading specifications, the time requirement may be too strict to allow the vehicle to perform additional manoeuvres in order to achieve the desired heading. The LPM must then make some trade-off, based on manually-assigned importance weightings for each parameter, between how accurately the goal position is achieved and how accurately the goal heading is achieved.

A trade-off between position and heading accuracy may be required in some cases, if the nature of the vehicle's mission requires heading to be specified, but many other

vehicle states may simply be disregarded in all cases. For example, it is rarely important to specify velocity or acceleration, and, even for surveillance purposes, pitch and roll become irrelevant if the vehicle has an orientable on-board camera.

In most cases, we can write the reduced state vector simply as

$$\mathbf{x}_r = \begin{bmatrix} \mathbf{p} \\ \psi \end{bmatrix}, \quad (4.3.1)$$

where \mathbf{p} denotes position in three dimensions and ψ specifies heading, the latter which may optionally be discarded from the reduced state vector depending on the vehicle application. In the remainder of this chapter, we will simply write \mathbf{x} to mean the reduced state (\mathbf{x}_r), unless specifically stated otherwise.

A method which connects an initial state-time (\mathbf{x}_0, t_0) to a local goal state-time (\mathbf{x}_g, t_g) directly and optimally, is called a Local Planning Method (LPM). For ideal Dubins curves, the LPM may make use of explicitly-defined position trajectories to calculate a solution. This is, however, not possible for manoeuvres consisting of the non-ideal primitives presented in Section 4.2.5. In this section, we present the implementation of an LPM which is able to construct manoeuvre-based solutions using an arbitrary, and implicitly-defined practical manoeuvre set, as captured using the method described in Section 4.2.5.2.

4.3.1 Local Path Planning as a Minimisation Problem

We aim to find a manoeuvre sequence, $\boldsymbol{\mu} = [m_1, m_2, \dots, m_n]^T$, and corresponding execution times for the constituent manoeuvres of \mathbf{m} , $\mathbf{t} = [\tau_1, \tau_2, \dots, \tau_n]^T$, which transitions the vehicle state from initial state \mathbf{x}_0 to local goal state \mathbf{x}_f , where $\mathbf{m} \in \mathcal{C}_M$, the collection of allowed manoeuvre sequences. We first consider the problem of finding suitable execution times for a particular manoeuvre sequence. The selection of the optimal manoeuvre sequence for a particular problem is addressed in Section 4.3.4.

Stated as a minimisation problem, we aim to find \mathbf{t} such that the error or *residual*, \mathbf{r} , between \mathbf{x}_f and the resultant state after applying the manoeuvre sequence transition function $M(\mathbf{t})$ on the vehicle, initially at \mathbf{x}_0 , is minimum.

Formally, we define the residual

$$\mathbf{r}(\mathbf{t}) = \mathbf{x}_f - M(\mathbf{x}_0, \mathbf{t}), \quad (4.3.2)$$

and aim to minimise the sum of squares of residuals

$$S(\mathbf{t}) = \mathbf{r}^T(\mathbf{t})\mathbf{r}(\mathbf{t}), \quad (4.3.3)$$

subject to the implicit restrictions

$$t_f = \sum_{i=1}^n \tau_i + t_0, \text{ and} \\ \tau_1, \tau_2, \dots, \tau_n \geq 0.$$

The sum of squares of the residuals is a quantitative representation of the error between the actual observed state after executing a specific sequence of manoeuvres and the desired final state.

4.3.2 Selecting an Appropriate Minimisation Algorithm

In order to minimise the sum of squares of residuals function presented as Equation 4.3.3, we require a reliable and fast minimisation algorithm.

We consider several common-used classes of minimisation algorithms:

- Grid minimum search,
- Downhill simplex (Nelder-Mead),
- Simulated Annealing,
- Nonlinear Conjugate Gradient,
- Quasi-Newton methods (e.g. BFGS), and
- Quasi-Gauss-Newton methods (e.g. Levenberg-Marquardt)

Grid minimum search is eliminated, as we know that the LPM operates in many dimensions and that the complexity of a grid-based search grows exponentially with each added dimension, making such algorithms computationally intractable.

Downhill simplex, also known as the Nelder-Mead method [32] (NMM), uses a simplex – that is, a geometric structure with $n + 1$ vertices in an n -dimensional space – with each vertex representing a sample of the function with certain parameters. This algorithm takes a sequence of steps, in each of which a vertex of the simplex with the highest cost is reflected through the centroid of the other vertices to obtain a new test point. If the cost at the test point is lower than the current best cost, the simplex is stretched across the sample, otherwise the size of the simplex is reduced [32]. The NMM is robust, but not very efficient and may converge to non-stationary values of a cost function [33].

Simulated Annealing (SA) [34; 35] is derived from a metallurgical technique used to increase crystal size by temperature treating a material. SA is able to find the global minimum of a function which may have many local minima. In each iteration, the algorithm randomly selects a solution from a candidate distribution [34; 35]. The new solution is either rejected or accepted based on a probability calculated using the difference between the new solution and the current best solution, as well as on a parameter referred to as the *temperature*, which is decreased as the algorithm execution time increases. This allows the algorithm to initially explore a large number of solutions very quickly, some potentially containing steps which follow a nondecreasing gradient, but to prefer steps which are along a decreasing gradient as execution time increases [34; 35]. SA regularly requires *restarts*, in which it reverts to an old solution that is significantly better than the current best parameter estimate [34; 35]. This makes it very inefficient for a large number of parameters and it is therefore not considered a good candidate for our problem.

Nonlinear Conjugate Gradient [36] uses the function gradient alone to find a minimum, which may not be global. The algorithm iteratively calculates the direction of steepest gradient descent, similar to the general method of steepest descent, but with an adjustable step length. This step length may be updated using one of several techniques. This algorithm is disregarded, as it is generally only suitable for linear and quadratic

functions and because it is known to be slower than quasi-Newton algorithms, such as BFGS, for most functions.

Quasi-Newton methods – that is, algorithms derived from Newton’s method to find the stationary point of a function – use the first *and* second derivatives of a function. For functions in higher dimensions, the Hessian matrix of second derivatives is used or, in some algorithms such as variations on the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method, the Jacobian is used instead in order to find the root of functions, rather than the gradient [37; 38; 39; 40]. Many variations on the method with subtly different update steps exist, including Symmetric Rank-One, Broyden, BFGS, and Davidson-Fletcher-Powell. BFGS is very popular and is efficient even with large numbers of variables, but it is usually outperformed by the Levenberg-Marquardt Algorithm (LMA) [41], especially for functions which are not well approximated by a quadratic curve. This is indeed the case for the functions typically encountered by our LPM (refer to Section 4.3.3.1).

In the remainder of this section, we provide a brief history of Quasi-Gauss-Newton algorithms from simple gradient evaluation to modern extensions of the Levenberg-Marquardt Algorithm (LMA) [42; 43], in order to motivate that LMA – specifically Fletcher’s version of LMA [44], is well-suited to our problem.

4.3.2.1 Gradient Evaluation Algorithms

Simple gradient evaluations methods, such as the steepest gradient descent algorithm, minimise the residual by iteratively finding the direction of steepest gradient along the function to minimise $S(\mathbf{t})$, and taking a step in that direction. The step size is determined by a constant β and the gradient at the previous point. The update step for this method after the k -th iteration is given by

$$\mathbf{t}_{k+1} = \mathbf{t}_k - \beta \mathbf{g}(\mathbf{t}_k), \quad (4.3.4)$$

where $\mathbf{g}(\mathbf{t}_k)$ is the gradient at the current iteration given by

$$\mathbf{g}(\mathbf{t}_k) = \nabla S(\mathbf{t}_k). \quad (4.3.5)$$

Gradient evaluation is very stable for small values of β , but the algorithm converges extremely slowly if β is too small [45]. If no information is available as to the possible range of gradients in the function to be minimised, there is no way to determine a sensible value for β .

4.3.2.2 Hessian Evaluation Algorithms

Hessian evaluation methods use a matrix of second derivatives instead of an arbitrary constant to scale the update step. This usually leads to a significantly faster rate of convergence compared to gradient evaluation algorithms. A commonly-used Hessian evaluation algorithm is Newton’s method (also referred to as the Newton-Raphson algorithm).

The update step for this algorithm is

$$\mathbf{t}_{k+1} = \mathbf{t}_k - \mathbf{H}^{-1}(S(\mathbf{t}_k))\mathbf{g}(\mathbf{t}_k), \quad (4.3.6)$$

where $\mathbf{H}(S(\mathbf{t}_k))$ is the Hessian matrix of $S(\mathbf{t}_k)$ and is given by

$$\mathbf{H}(S) = \begin{bmatrix} \frac{\partial^2 S}{\partial x_1^2} & \frac{\partial^2 S}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 S}{\partial x_1 \partial x_m} \\ \frac{\partial^2 S}{\partial x_2 \partial x_1} & \frac{\partial^2 S}{\partial x_2^2} & \cdots & \frac{\partial^2 S}{\partial x_2 \partial x_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 S}{\partial x_n \partial x_1} & \frac{\partial^2 S}{\partial x_m \partial x_2} & \cdots & \frac{\partial^2 S}{\partial x_m^2} \end{bmatrix}. \quad (4.3.7)$$

Comparing Equation 4.3.6 to Equation 4.3.4, we see that the inverted Hessian fulfils the same role as β , but it provides better matched step sizes to expedite the solution [45]. Calculation of the Hessian is computationally complex in higher dimensions and some algorithms, such as the Gauss-Newton Algorithm (GNA), therefore use the Jacobian matrix $\mathbf{J}(S)$ instead, where $J(S)$ is a matrix of partial derivatives of the constituent residual functions of S :

$$\mathbf{J}(S) = \begin{bmatrix} \frac{\partial r_1}{\partial x_1} & \frac{\partial r_1}{\partial x_2} & \cdots & \frac{\partial r_1}{\partial x_m} \\ \frac{\partial r_2}{\partial x_1} & \frac{\partial r_2}{\partial x_2} & \cdots & \frac{\partial r_2}{\partial x_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial r_n}{\partial x_1} & \frac{\partial r_n}{\partial x_2} & \cdots & \frac{\partial r_n}{\partial x_m} \end{bmatrix}. \quad (4.3.8)$$

Assuming a small residual, the Hessian is then approximated by [46]

$$\mathbf{H} \simeq \mathbf{J}^T \mathbf{J}. \quad (4.3.9)$$

Furthermore it can be shown that the gradient vector \mathbf{g} can be approximated by [45]

$$\mathbf{g} \simeq \mathbf{J}\mathbf{r}, \quad (4.3.10)$$

where \mathbf{r} is a vector of n residuals in m parameters with $n \geq m$:

$$\mathbf{r} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_n \end{bmatrix}. \quad (4.3.11)$$

Combining the update step of Newton's method in Equation 4.3.6 with Equations 4.3.9 and 4.3.10, we obtain the update step for the GNA,

$$\mathbf{t}_{k+1} = \mathbf{t}_k - [\mathbf{J}_k^T \mathbf{J}_k]^{-1} \mathbf{J}_k \mathbf{r}_k, \quad (4.3.12)$$

with

\mathbf{J}_k meaning $\mathbf{J}(S(\mathbf{t}_k))$ and
 \mathbf{r}_k meaning $\mathbf{r}(\mathbf{t}_k)$.

GNA is computationally simpler than Newton's original algorithm, as it does not require the evaluation of second derivatives, and both these algorithms are significantly faster than gradient evaluation in most applications. Unlike gradient evaluation algorithms, however, this class of algorithm may diverge, as the matrix $\mathbf{J}_k^T \mathbf{J}_k$ is not guaranteed to be invertible [45].

4.3.2.3 Hybrid Algorithms

Hybrid algorithms, based on the pioneering work of Levenberg [42] and Marquardt [43], represent a compromise between speed and reliable convergence.

The update step in Levenberg's original algorithm is very similar to that of the GNA, but an additional precaution is taken to ensure that $\mathbf{J}_k^T \mathbf{J}_k$ in Equation 4.3.9 is always invertible. This is done by adding a diagonal of positive constants to the aforementioned matrix. The Hessian is now approximated as

$$\mathbf{H} \simeq \mathbf{J}^T \mathbf{J} + \lambda \mathbf{I}, \quad (4.3.13)$$

with

$$\lambda \geq 0.$$

By substituting Equation 4.3.13 into the update step for the GNA in Equation 4.3.12, we find the update step for the Levenberg algorithm [42]:

$$\mathbf{t}_{k+1} = \mathbf{t}_k - [\mathbf{J}_k^T \mathbf{J}_k + \lambda \mathbf{I}]^{-1} \mathbf{J}_k \mathbf{r}_k, \quad (4.3.14)$$

with *combination coefficient*

$$\lambda \geq 0.$$

It can be inferred that for a large λ , Equation 4.3.14 is approximately equivalent to Equation 4.3.4 and, for values of λ close to zero, it is approximately equivalent to Equation 4.3.12. The method of Levenberg therefore becomes equivalent to the method of steepest descent when $\lambda \rightarrow \infty$ and approaches the GNA when $\lambda \rightarrow 0$.

Marquardt modified the Levenberg algorithm by replacing the identity matrix in Equation 4.3.14 with a diagonal matrix containing the diagonal elements of $\mathbf{J}_k^T \mathbf{J}_k$ in order to provide damping that is scaled appropriately in every dimension. This revised version of the algorithm is known as the Levenberg-Marquardt Algorithm (LMA) and its update step can be expressed as [43]

$$\mathbf{t}_{k+1} = \mathbf{t}_k - [\mathbf{J}_k^T \mathbf{J}_k + \lambda \text{diag}(\mathbf{J}_k^T \mathbf{J}_k)]^{-1} \mathbf{J}_k \mathbf{r}_k. \quad (4.3.15)$$

For convenience, we will henceforth represent the update term in Equation 4.3.15 by the symbol δ_k . That is to say,

$$\delta_k \triangleq -[\mathbf{J}_k^T \mathbf{J}_k + \lambda \text{diag}(\mathbf{J}_k^T \mathbf{J}_k)]^{-1} \mathbf{J}_k \mathbf{r}_k. \quad (4.3.16)$$

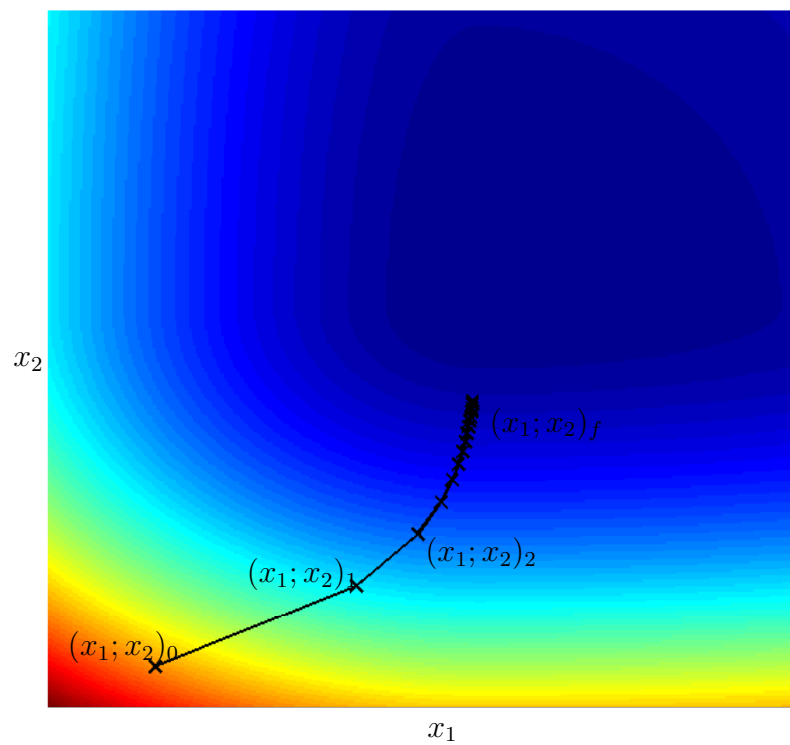


Figure 4.5 – Levenberg-Marquardt minimisation traced across several iterations.

In Marquardt's implementation, λ is initially chosen arbitrarily and is then updated according to the following iterative process [43]:

1. Decrease λ by constant factor v .
2. Do an update as per Equation 4.3.15 to obtain a new parameter estimate \mathbf{t}_{k+1} .
3. Evaluate the sum of squares of residuals, S_{k+1} at \mathbf{t}_{k+1} .
4. If $S_{k+1} > S_k$, discard the update and increase λ by constant factor v , then go to Step 2.
5. If $S_{k+1} \leq S_k$, accept the update and go to Step 1.

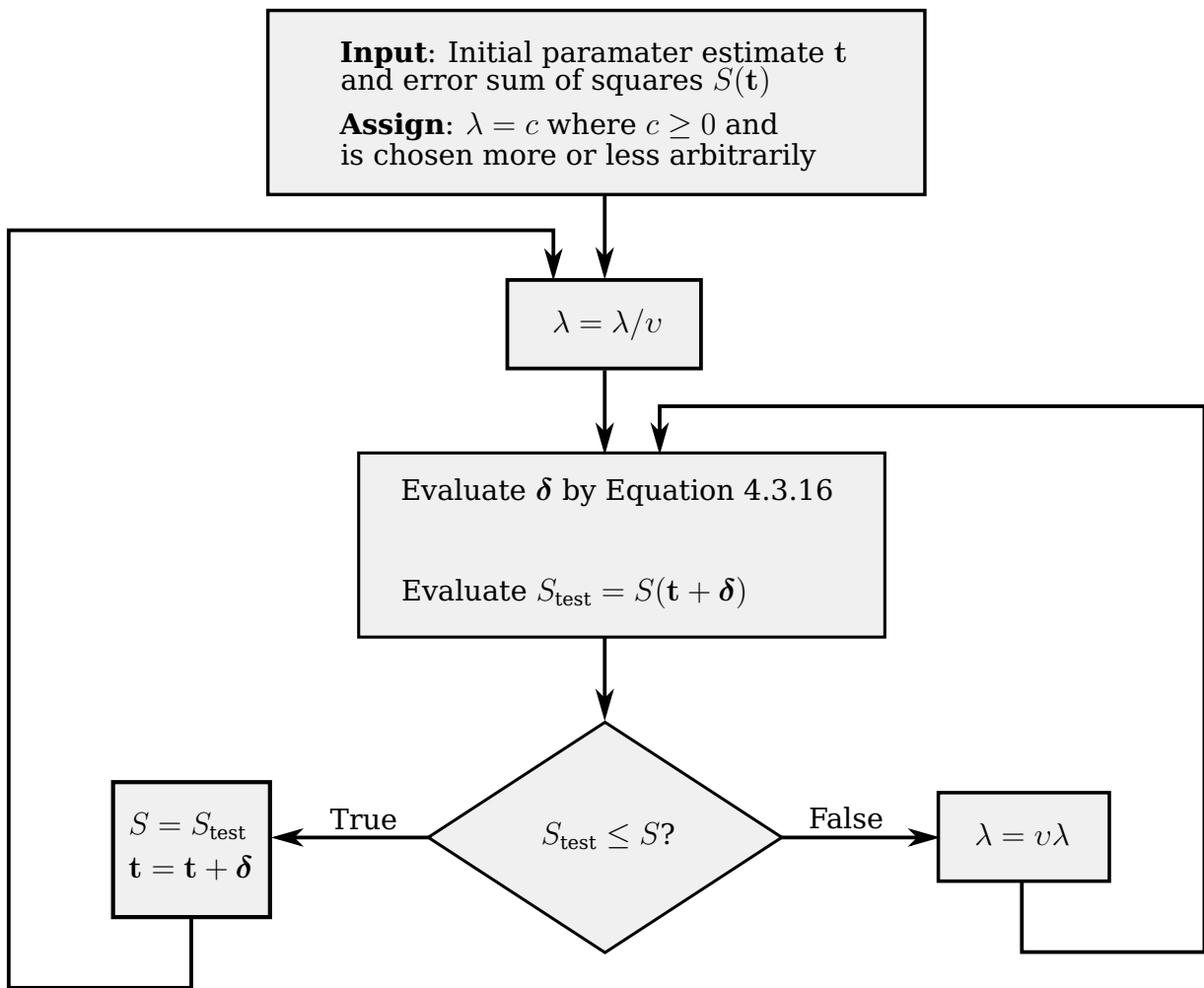


Figure 4.6 – Update process for the combination coefficient λ in the original Levenberg-Marquardt algorithm [44]. The algorithm terminates (not shown) when δ falls below a pre-defined cutoff value.

This process, which is illustrated in Figure 4.6, allows the LMA to switch seamlessly between the method of steepest descent and the GNA as necessary. This means that the LMA provides the stability of gradient descent algorithms, yet converges much more

rapidly, although not as fast as the pure Hessian evaluation algorithms. This rapid convergence is illustrated in Figure 4.5 by the visual trace of the LMA for an arbitrary function.

The LMA does, however, have certain shortcomings. Fletcher highlights the four most notable flaws [44]:

1. A poor initial choice of λ wastes many iterations before the LMA updates this parameter to a more suitable value.
2. Decreasing λ by factor v on the first iteration is arbitrary and may be excessive for a large v .
3. The test $S_{k+1} \leq S_k$ in the LMA precludes proof that the parameter estimate is converging to a minimum.
4. The LMA only achieves superlinear convergence, whereas the GNA can converge quadratically.

In order to overcome the shortcomings of the LMA, Fletcher introduces an improved updating mechanism for the parameter λ . Instead of arbitrarily choosing and scaling λ , a suitable value for λ is calculated based on the difference between the actual change in S and the change predicted in each iteration using a linear model [44].

The predicted improvement in S at iteration k is given by [44]

$$\Phi(\mathbf{t}_k) - \Phi(\mathbf{t}_k + \boldsymbol{\delta}_k) = -2\boldsymbol{\delta}_k^T \mathbf{J}_k^T \mathbf{r}_k - \boldsymbol{\delta}_k^T \mathbf{J}_k^T \mathbf{J}_k \boldsymbol{\delta}_k, \quad (4.3.17)$$

where

Φ represents a predicted value, as per Fletcher's [44] notation.

The ratio of actual to predicted reduction in S is given by

$$R = \left| \frac{\mathbf{t}_k - (\mathbf{t}_k + \boldsymbol{\delta}_k)}{\Phi(\mathbf{t}_k) - \Phi(\mathbf{t}_k + \boldsymbol{\delta}_k)} \right|, \quad (4.3.18)$$

where the denominator, $\Phi(\mathbf{t}_k) - \Phi(\mathbf{t}_k + \boldsymbol{\delta}_k)$, is calculated as per Equation 4.3.17.

If the ratio R is close to unity, λ is reduced and if the ratio is close to 0, it is increased. For some intermediate range of values for R , λ is left unaltered. Fletcher defines two constants, $\rho = 0.25$ and $\sigma = 0.75$, as the minimum and maximum cutoffs for this intermediate range, respectively. It is shown that the algorithm is largely insensitive to the exact values of these parameters [44; 47].

In order to increase λ , Fletcher keeps the geometric progression method used by the LMA, but allows the scaling factor v to vary between fixed lower and upper bounds.

Specifically [44]

$$v = 2 - (S_{k+1} - S_k) / \delta^T \mathbf{J}_k^T \mathbf{r}_k, \quad (4.3.19)$$

with⁴

$$2 \leq v \leq 10.$$

When decreasing λ on the other hand, Fletcher argues that a geometric reduction in λ is bad since λ can never reach zero, thus never resulting in true quadratic convergence [44]. In order to circumvent this, a cut-off value is introduced for λ , below which its value is set to zero. This cut-off value is given by [44]

$$\lambda_c = \frac{1}{\left\| [\mathbf{J}_k^T \mathbf{J}]^{-1} \right\|}. \quad (4.3.20)$$

The value of λ_c is recalculated whenever λ is increased from zero.

Fletcher's version of the LMA (FLMA) does therefore not rely on an arbitrary initial estimate of λ , and instead calculates reasonable values for this parameter to effect faster rates of convergence than standard LMA without sacrificing stability.

Due to the combination of reliability and speed offered by FLMA, it is the method of choice for our LPM. In the next section, we provide motivation that the FLMA is suitable for the type of functions generated by our LPM.

4.3.3 Applying Fletcher's Version of the Levenberg-Marquardt Algorithm to the Local Planning Problem

The minimisation algorithm selected for use in the LPM is Fletcher's version of the Levenberg-Marquardt Algorithm (FLMA) [44]. Before implementing this algorithm in our LPM, we must first ensure that the functions the LPM will encounter are well-behaved in terms of the limitations of FLMA.

The use of the FLMA implies that the cost function to be minimised must have a continuous, non-zero numerical second derivative. If linear interpolation is used in reconstructing manoeuvres from discretised data, the FLMA may terminate prematurely on false stationary points and fail to find a solution. This is the reason for the use of second order interpolation, as was mentioned, but not adequately explained, in Section 4.2.5.2.

4.3.3.1 Convergence

As the FLMA tests convergence in each iteration by evaluating $S_{k+1} \leq S_k$, it is a sufficient, but not necessary, condition that the function to which it is applied be of a form such that, at each iteration where the parameter estimate is closer to the global minimum, the value of S is nonincreasing. If our function were therefore strictly convex, it would be possible to guarantee convergence in all cases.

It must be noted, however, that the FLMA is able to find the global minimum for a function even if it is not strictly convex, as long as the algorithm is able to iterate through

⁴Fletcher suggests the restriction $2 \leq v \leq 10$, based on intuitive arguments and empiric results [44].

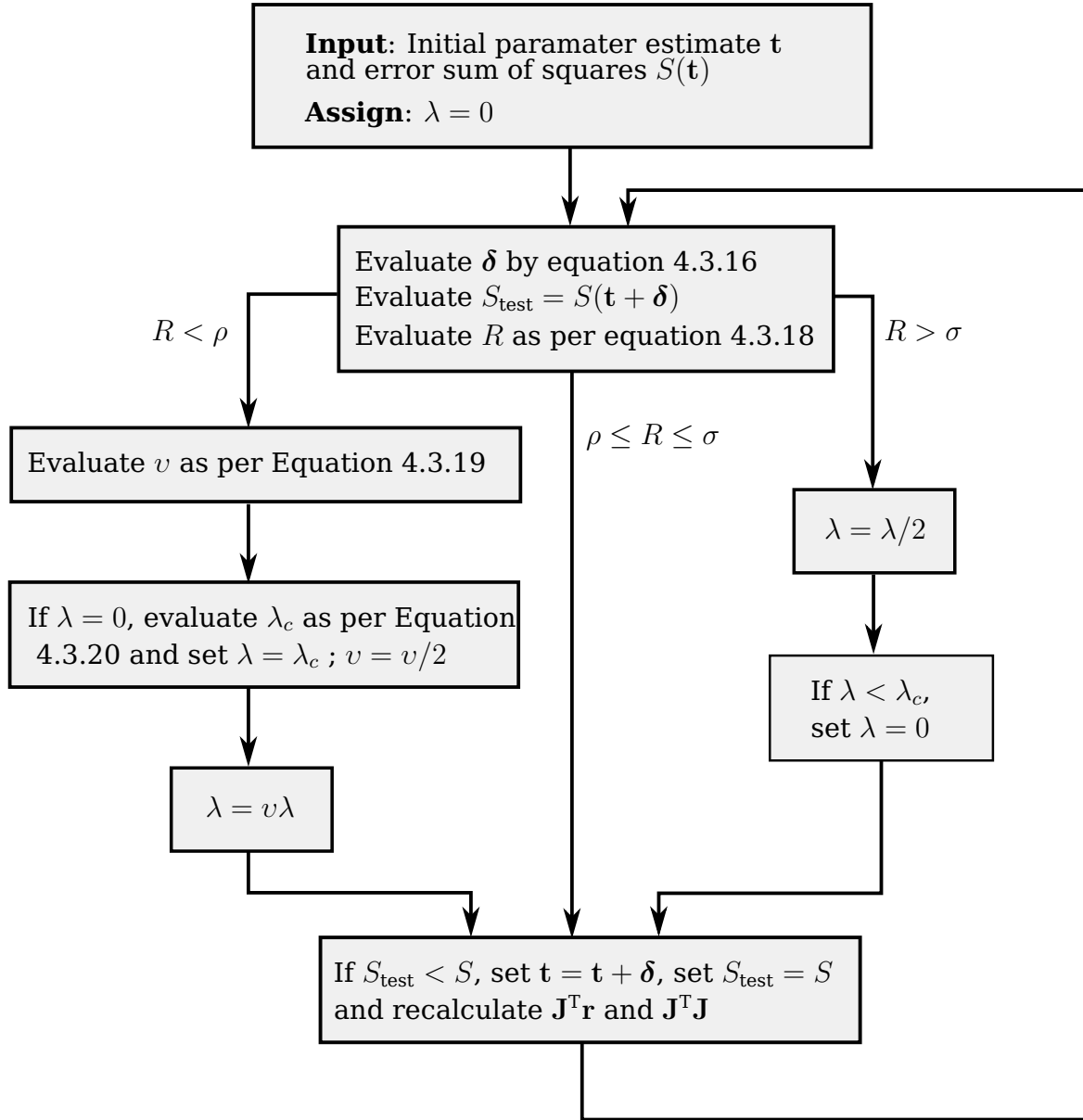


Figure 4.7 – Update process for the combination coefficient λ in Fletcher’s version of the Levenberg-Marquardt algorithm [44]. The algorithm terminates (not shown) when δ falls below a predefined cutoff value.

a sequence of discrete steps such that the value of S is nonincreasing between each pair of points in this sequence. Stated differently, if there exists a path from the initial estimate to the global minimum which is nonincreasing when discretised by the smallest allowable step size of the FLMA, the global minimum can be found in many cases [44; 47; 48]. In order to get an idea of the form of a typical function that may be encountered in our LPM, let us consider a simple residual defined only in terms of position and time,

$$\mathbf{r} = \begin{bmatrix} \mathbf{p}_f \\ \mathbf{t}_f \end{bmatrix} - \begin{bmatrix} \mathbf{p}'_f \\ \mathbf{t}'_f \end{bmatrix}. \quad (4.3.21)$$

Let us first consider the sum of squares function for an ideal Dubins turn-straight-turn manoeuvre sequence.

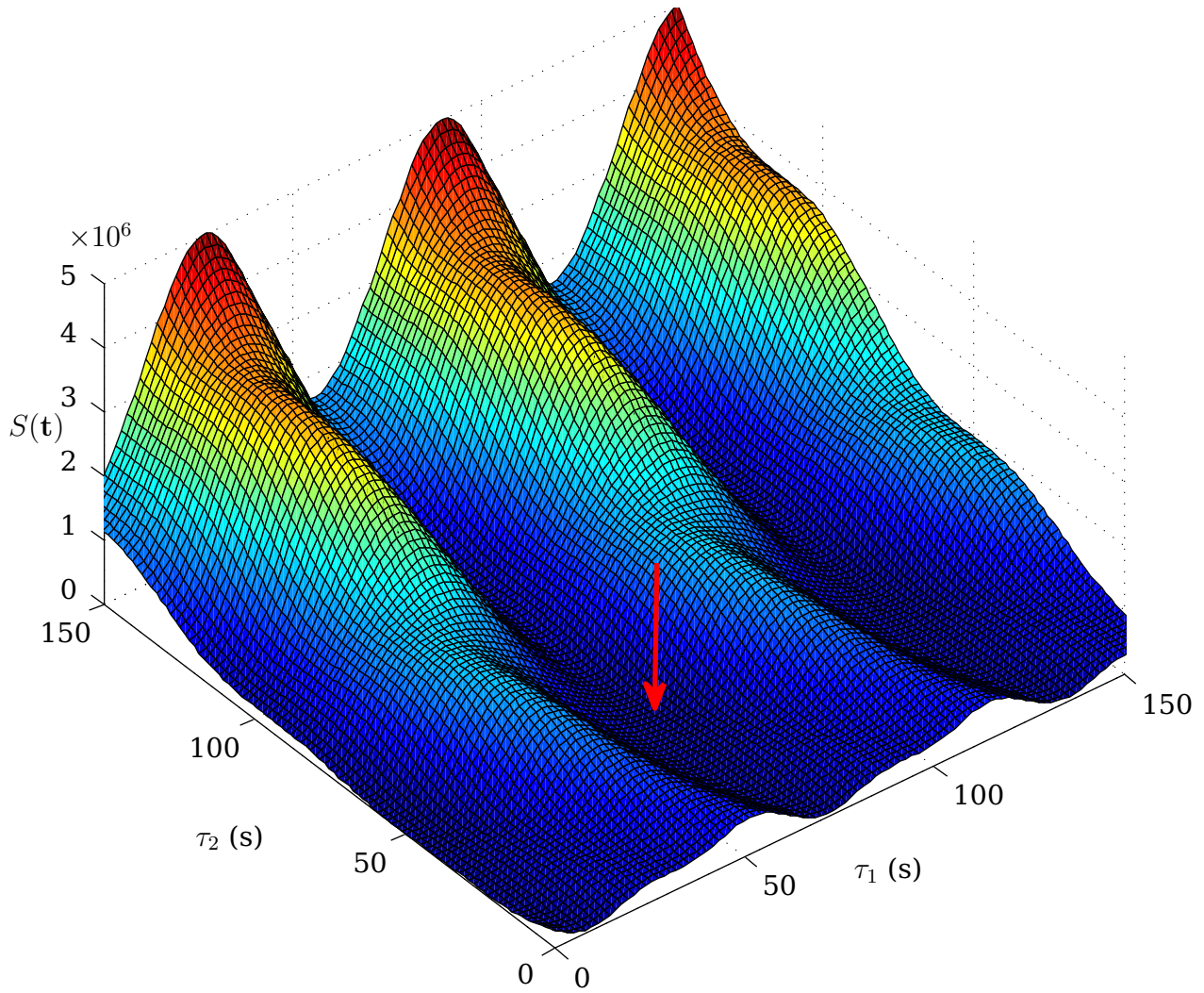


Figure 4.8 – Typical relationship between manoeuvre sequence execution times and the residual-sum-of-squares cost for a turn-straight-turn manoeuvre. The optimal solution is indicated by the red arrow. The specified target final time for the example shown is 160s.

It can be shown that for this case

$$\begin{aligned}
 S = \mathbf{r}^T \mathbf{r} = & \left[2\rho \sin\left(\frac{v}{\rho}\tau_1\right) + v\tau_2 \cos\left(\frac{v}{\rho}\tau_1\right) + \rho \sin\left(\frac{v}{\rho}(t_f - \tau_2 - 2\tau_1)\right) - p_{x_f} \right]^2 \\
 & + \left[\rho - 2\rho \cos\left(\frac{v}{\rho}\tau_1\right) + v\tau_2 \sin\left(\frac{v}{\rho}\tau_1\right) + \rho \cos\left(\frac{v}{\rho}(t_f - \tau_2 - 2\tau_1)\right) - p_{y_f} \right]^2 \\
 & + [ut_f - p_{z_f}]^2,
 \end{aligned} \quad (4.3.22)$$

where

- τ_n is the variable n -th execution time estimate,
- p_{d_f} is the specified final position on axis d ,
- ρ is the constant minimum turning radius of the vehicle,
- v is the constant horizontal velocity component and
- u is the equivalent constant vertical velocity as defined in Section 4.2.3.

The projection in (τ_1, τ_2, S) of Equation 4.3.22 with parameters substituted from an example problem using a turn-straight-turn manoeuvre sequence, is shown in Figure 4.8. Note that τ_3 can be found trivially for any data point by solving for it in the relationship

$$t_f = \tau_1 + \tau_2 + \tau_3. \quad (4.3.23)$$

The function is seen to consist of periodically repeating trenches along the τ_1 -axis. These trenches occur due to the periodic nature of the turn primitives at the beginning and end of the Dubins curve manoeuvre sequence. The period at which the trenches repeat is thus equal to the time the vehicle takes to complete one full revolution in the horizontal plane,

$$T_{\text{trench}} = \frac{2\pi\rho}{v}. \quad (4.3.24)$$

The base of each trench slopes gently downward and, after reaching the minimum, upward along the τ_2 -axis. This can be attributed to the straight line section of the Dubins curve gradually approaching and then overshooting the goal state as the corresponding execution time, τ_2 increases.

Our function is clearly not strictly convex. It is, however, known that the FLMA is able to find a global minimum for some non-convex functions containing such trenches. In fact, the original benchmark used by Fletcher to evaluate performance of the FLMA is the Rosenbrock sum of squares function [44], which is given by [49]

$$f(\mathbf{x}) = \sum_{i=1}^{N-1} [(1 - x_i)^2 + 100(x_{i+1} - x_1^2)^2], \forall x \in \mathbb{R}^N. \quad (4.3.25)$$

Our function bears a strong resemblance to the Rosenbrock function [49], as is evident when comparing the form of Equation 4.3.25 to that of Equation 4.3.22.

Based on Fletcher's finding that the FLMA is able to reach the minimum of the Rosenbrock function in 3 to 15 iterations and to an accuracy of 5×10^{-4} in each variable [44], we have evidence to suggest that the FLMA will likely converge reliably and in an acceptable number of iterations for our problem, provided that our initial parameter estimate lies within the correct "trench" of our function.

The global minimum for some specific problem is indicated by a red arrow in Figure 4.8. The trench in which this minimum lies could vary for the same problem: if the goal time is specified to be much longer than the minimum time required to reach the goal position, different trenches represent a different number of "loops" during the first and last primitives (in the horizontal plane) executed by the vehicle to bide time. This is not necessarily a bad solution to bidding time, as long as the turn primitive is defined such that it represents an efficient holding pattern for our vehicle when executed as a complete revolution. If, on the other hand, we wish to eliminate this behaviour, we could explicitly prevent multiple revolutions by artificially limiting the length of the turn components in every manoeuvre sequence, although doing so yields no obvious benefit. If we allow multiple revolutions, it is not trivial to determine whether these revolutions are preferable during the first or last motion primitive: arguments can be made for both options based on very specific environments.

For example, if we know that access to the goal position will become sealed off imminently, it is better to reach the goal position using the shortest path length and to subsequently execute a holding pattern about it until the desired goal time elapses. If, on the other hand, the vehicle is initially sealed-off from the goal position and is only able to access the goal later, it is desirable to execute the holding pattern first instead. For our purposes, we assume that this kind of information is not available and we therefore have no preference for a solution which lies in a specific trench of the cost function; the only requirement is that our total execution time is correct.

With this consideration, the FLMA should be suitable to minimise our cost function, at least in the dimensions we have considered. Minimisation in the z -position dimension is, however, trivial, as we consider vertical motion to be independent of horizontal motion and, as we can clearly see from Equation 4.3.22, our cost function does not exhibit periodic behaviour in the third spatial dimension.

Provided that the practical manoeuvres for our vehicle do not drastically alter the shape of the cost function, the arguments made in this section remain applicable.

4.3.3.2 Computational Complexity

Time complexity for both LMA and FLMA is dominated by the matrix inversion required in each iteration (refer to Equation 4.3.15). This leads to $\mathcal{O}(n^3)$ time complexity and $\mathcal{O}(n^2)$ storage complexity [50].

If we define residuals in three spatial dimensions, time and heading, the FLMA requires approximately

$$x = 5^3 = 125 \text{ FLOP per iteration.} \quad (4.3.26)$$

Using Fletcher’s observation that at most 15 iterations of the FLMA are required to find the minimum of a sum of squares function [44], we should, in theory, require at most

$$X = 125 \times 15 = 1875 \text{ FLOP per minimisation.} \quad (4.3.27)$$

Since most modern processors can achieve in excess of 1 GFLOPS, the FLMA should theoretically be able to find the optimal parameter estimate in under $2\mu\text{s}$ ⁵. Computational complexity is therefore not a major concern.

For the purposes of our project we develop a custom implementation of FLMA in ISO/IEC 9899:1999 specification C. Our code is based in part on an unofficial MATLAB implementation by Balda [48].

4.3.4 Manoeuvre Sequence Selection

Once an acceptable method to calculate the optimal motion primitive execution times for a specific manoeuvre sequence is in place, we require a means by which the LPM can determine the optimal manoeuvre sequence to connect two states.

A simple approach is to try every sequence contained in the manoeuvre sequence collection $\mathcal{C}_{\mathcal{M}}$. Based on our calculations in Section 4.3.3.2, the FLMA is extremely fast, which implies that testing every sequence should be a relatively inexpensive procedure if $\mathcal{C}_{\mathcal{M}}$ does not contain many elements. Indeed, it is argued in Section 4.2.3 that only six manoeuvre sequences are necessary. A more complex approach to this problem is therefore not justified.

⁵In practice, the execution time is expected to be significantly longer due to factors such as memory access time and function call overhead. The point of this calculation is to illustrate that the algorithm itself is fast enough to be feasibly implemented in an LPM which may call it thousands of times per second.

Chapter 5

System Evaluation

5.1 Overview

In Section 1.3, several objectives were set for the motion planning algorithm developed in this project. In this chapter, we define several test environments to verify that all the criteria for the motion planning algorithm have been fulfilled.

Additionally, we analyse the contributions of the various constituent parts of the algorithm towards total processing time, in order to provide recommendations for more efficient implementation.

5.2 Test Environment

The outer loop of the PRM-algorithm is implemented mainly in Matlab script, while the PRM graph structure and LPM are implemented in ISO/IEC 9899:1999 specification C, compiled into an x86-compatible DLL.

All tests presented in this chapter are performed on a desktop computer with an Intel i5 2500K quad-core CPU, running at 3.42GHz, with 32KB L1, 256KB L2, and 6144KB L3 cache per core. The computer is equipped with 2×4GB DDR3-667 RAM modules.

The processor performance is gauged using the Intel LINPACK benchmark [51], on which it is found to score 86.7GFLOPS with one thread per core, or 23.6GFLOPS using only a single thread with affinity to a specific core. As our code utilises only a single core, the latter benchmark is of more relevance.

We prefer to express execution performance in seconds rather than number of operations throughout this chapter, in conformance with publications stemming from similar research [1; 11; 14; 2].

5.3 Comparing Ideal and Practical Manoeuvre Sets

In Sections 4.2.4 and 4.2.5, two approaches to defining manoeuvres were described. The first and most common approach defines geometrically-perfect three-dimensional Dubins curves. For such ideal manoeuvres, a complex control technique, for example Specific Acceleration Matching (SAM), is required to facilitate accurate path-following [5]. It was hypothesised in Section 4.2.5 that this technique may be unnecessarily energy-intensive,

since even a relatively large deviation from the nominal path may be insignificant compared with uncertainties in predicting conflict with objects in the environment.

As an alternative approach, it was suggested that this complex control system be replaced by a simple control system which does not control lateral acceleration directly, but rather indirectly via a constant banking angle during turns and a zero banking angle for straight and level flight. For this approach, the nominal trajectories of specific manoeuvres are determined empirically by simulating the vehicle with its controllers and recording the path under ideal conditions. This recorded data is made available off-line for use in the LPM of the motion planning algorithm. In Sections 4.2.4 and 4.2.5, it was hypothesised that this control approach requires less energy than path-following techniques such as those based on SAM.

5.3.1 Energy Efficiency

The major anticipated benefit of using a simple control system as opposed to path following techniques is a reduction in the energy consumed in reaching the goal state. In order to compare the different control approaches in this regard, we require a means by which to quantify the energy expended by our vehicle.

5.3.1.1 Quantifying energy consumption

The test aircraft is propelled by a single Hyperion ZS 4025 10-turn brushless electrical motor [3].

Assuming that the back EMF produced in the motor is negligible under typical load, we can say that the electrical power consumed by the motor is directly proportional to the square of the applied voltage, or

$$P_e \propto V_{\text{in}}^2. \quad (5.3.1)$$

From this, it follows that the total electrical energy expended in propelling the vehicle during the time interval t_0 to t_f ,

$$E \propto \int_{t_0}^{t_f} V_{\text{in}}^2 dt. \quad (5.3.2)$$

The applied voltage, $V_{\text{in}}(t)$, is controlled by a Hyperion ATLAS Series Electronic Speed Controller (ESC) [3] which maps a digital numeric input linearly to a known output. Our controllers provide the input for this ESC via the throttle command ν , which is a known quantity provided by the control system. The linear mapping provided by the ESC, combined with Equation 5.3.2, imply that

$$E \propto \int \nu^2(t) dt. \quad (5.3.3)$$

For our discrete-time simulations, the right hand side of Equation 5.3.3 can be approximated by the Riemann sum

$$E = k_E \sum_{i=1}^n \nu_i^2(t) T_s, \quad (5.3.4)$$

where

T_s is the sampling period,

$$n = \frac{t_f - t_0}{T_s} \text{ and}$$

k_E is a constant of proportionality.

Equation 5.3.4 allows us to quantify the energy expended by our vehicle using only the throttle command provided by the control system. Since we aim only to compare relative energy consumed between the two different control and planning approaches, quantities proportional to energy consumed will suffice and we therefore do not need to determine the value of k_E .

5.3.1.2 Comparison of energy consumption using different control approaches

In order to compare the performance of the two control systems introduced in Section 4.2, a simulated test environment is created. This test environment is infinitely large and contains no obstructions or ground plane. The LPM is used to connect the initial position of the aircraft at (0; 0; -150) in the NED coordinate system to a goal position, which is sampled uniformly between 200m and 800m away. This is done for both geometrically-perfect manoeuvres, which are executed using the path-following control system, and non-ideal manoeuvres, using the simple control system.

Wind gusts along the inertial axes are modelled using band-limited white noise, the RMS of which can be adjusted. Constant winds are not simulated, as these can be detected through direct measurement or, indirectly, by measuring constant drift from the nominal path using data from the GPS and IMU. Once constant wind speed is known, it can be compensated for during the path-planning process by offsetting the goal position using a simple velocity-time calculation.

The wind components generated along the inertial axes are converted to body axis components through multiplication with the DCM. The maximum wind gust strength is chosen so as not to exceed a *fresh breeze*, defined on the Beaufort wind force scale as a maximum of 10.7ms^{-1} [52]. Wind gusts of this magnitude are considered barely-safe operating conditions for our aircraft, as its nominal airspeed is only 25ms^{-1} . Theoretically, the aircraft may lose lift or overturn if subjected to repeated adverse gusts which approach its airspeed, especially during an aggressive climb or banking manoeuvre.

Simulations are carried out with moderate wind gusts in the aforementioned test environment for one thousand valid position pairs which are successfully connected by the LPM. For each simulation, a pseudorandom seed, based on the Matlab CPU time, is selected for the wind model. The simulation is also repeated with no wind.

The results of this experiment are summarised in Table 5.1 and an example of a typical execution is shown in Figure 5.1.

The mean error in the final position relative to the desired goal state is observed to be slightly smaller for the simple control system with no wind, but significantly higher with strong gusts. This difference between the two control systems was observed mainly in the horizontal plane. The altitude error was, on the contrary, very similar for the two control systems. This can be attributed to the fact that climb rate is enforced directly by both control systems, while the simple control system affects lateral velocity only indirectly by controlling the banking angle.

Table 5.1 – Comparison of Path Following Control (PFC) and Simple Control (SC) performance for a test environment with wind gusts.

Metric	PFC	SC
Position Error, No Wind (%)	0.8	0.3
Position Error, Moderate Gusts (%)	1.2	7.8
Energy Consumption [†] , no Wind (%)	–	–1.9
Energy Consumption [†] , Moderate Gusts (%)	+6.8	–1.1

[†] Relative to energy consumed by the path-following control system under wind-free conditions.

It is important to note that the large position error for the simple control system under windy conditions can be reduced using a constant-replanning approach. This approach is not only beneficial in reducing the position error at every milestone, but a necessary precaution to cope with dynamic changes in the environment. Replanning and its effects are analysed in Section 5.4.2.

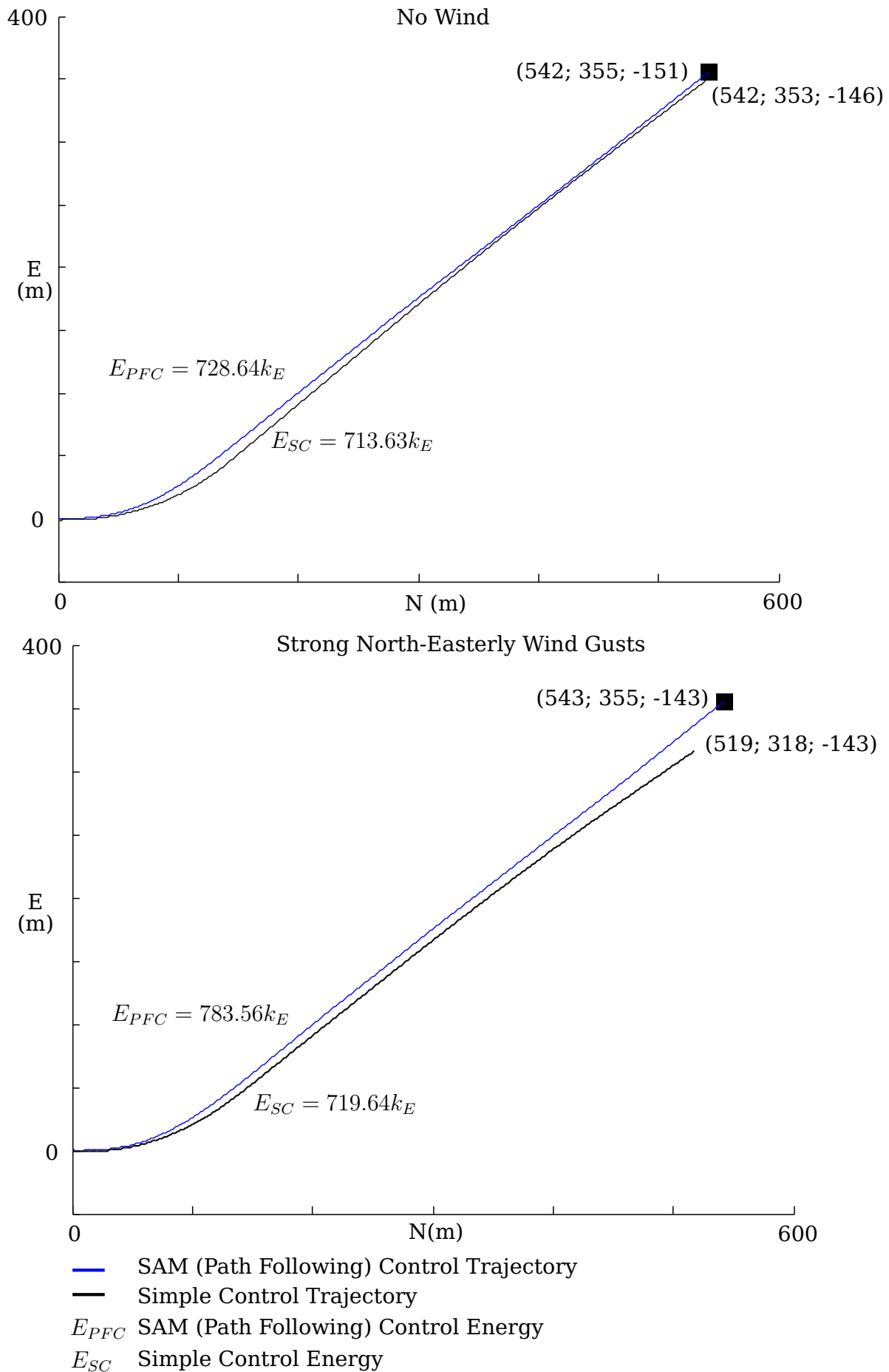


Figure 5.1 – Comparison of energy expended for different control systems. The symbol k_E is the proportional constant defined in Equation 5.3.4

5.4 Local Planning Method Evaluation

Before testing the LPM as an integral part of a larger motion planning algorithm, we aim to verify that it works reliably and efficiently as a stand-alone function.

5.4.1 Speed and Reliability

In order to gauge reliability and speed of the Local Planning Method developed in Chapter 4, we define an empty cubic environment with a side length of two thousand metres, and generate uniformly distributed state-time pairs in this space. Some of the pairs are connectible, given the dynamic constraints on our vehicle, and some are not. If we reject pairs with a velocity constraint test which is considerably more strict than required by the actual velocity constraint on the vehicle, we can be confident that the remaining points are connectible.

Initially, however, we generate one thousand state-time pairs with no rejection of points, and task the LPM with creating a connection between every pair. Just under 35% of the state-time pairs are connected successfully. The mean and standard deviation of the execution times are given by

$$\bar{t} = 84.4\text{ms and } \sigma = 84.9\text{ms.}$$

Another one thousand state-time pairs are generated, this time using rejection sampling as described above. The LPM is again applied to all pairs. In this experiment, 100% of the pairs are connected successfully. The mean and standard deviation of execution times for state-time pairs which are known to be connectible, are

$$\bar{t}_{\text{valid}} = 129.3\text{ms and } \sigma_{\text{valid}} = 64.4\text{ms.}$$

The distribution of execution times for this experiment is shown in Figure 5.2.

Comparing the distribution of execution times for the two cases, it becomes clear that the LPM terminates relatively quickly when no valid solution exists. As the larger motion planning algorithm applies rejection sampling before tasking the LPM to connect states, the vast majority of state-time pairs received by the LPM are expected to be connectible. The latter experiment (Figure 5.2) is therefore probably a better indication of practical performance of the LPM.

As is apparent from the distribution in Figure 5.2, approximately 95% of problems posed are solved by the LPM in under 260ms and only in a few, isolated cases does the LPM require more than 400ms to find a solution. This is relatively slow, compared with LPMs which can connect states directly by applying simple, closed-form algebraic expressions, but it is difficult to judge the influence of the increased execution time until the LPM is tested as an integral part of the larger motion planning algorithm. Experiments to verify performance of the motion planning algorithm as a whole, are detailed in Section 5.5.

5.4.2 Replanning

In Section 5.3.1.2, it was shown that the use of a control system which does not implement position tracking, leads to position errors in environments with strong disturbance

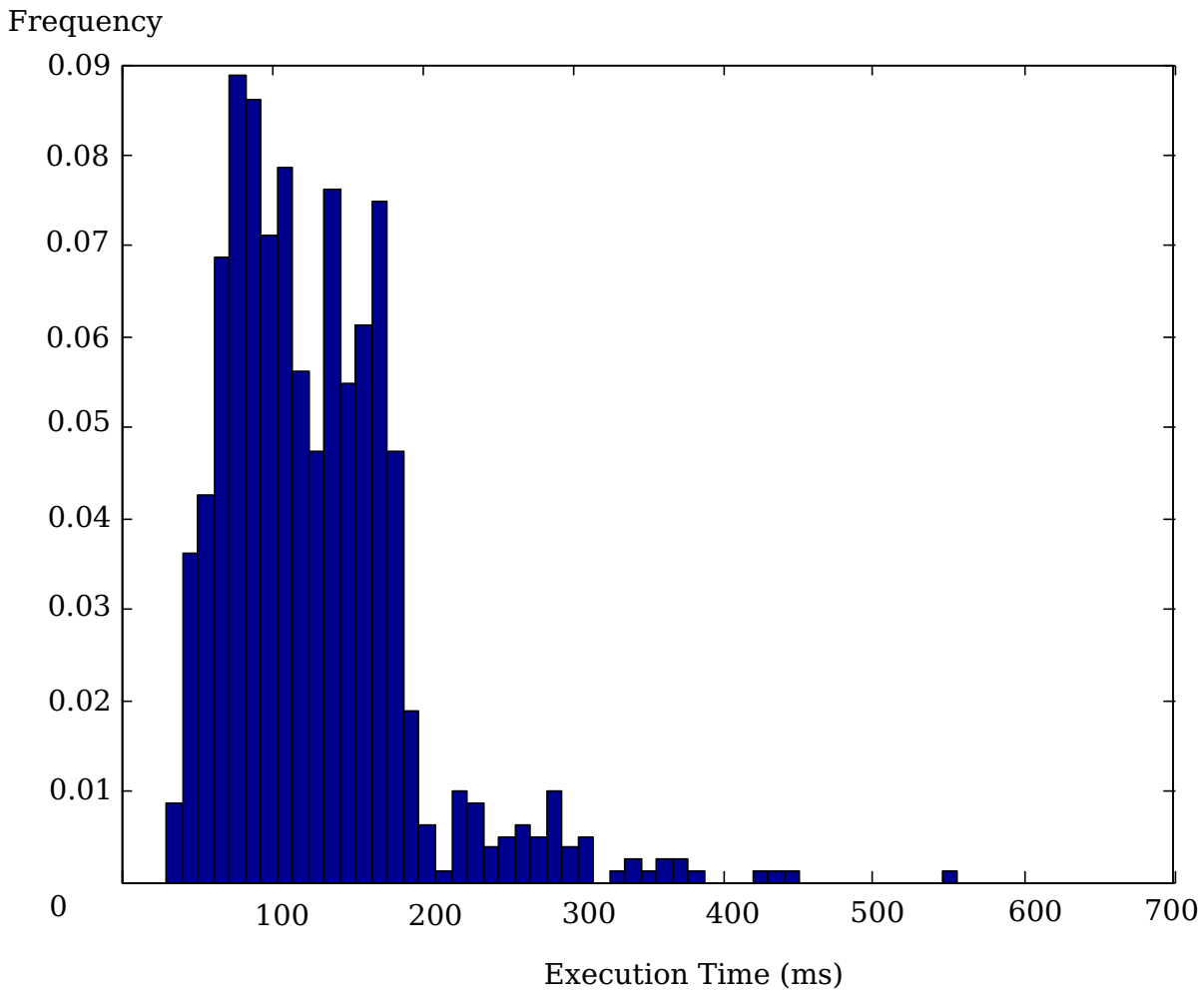


Figure 5.2 – Distribution of LPM Execution Times (1000 trials).

signals (e.g. wind gusts). It was hypothesised that these errors may be reduced through periodic replanning using the LPM.

In order to test this hypothesis, we revisit the windy environment described in Section 5.3.1.2. In Figure 5.3, the original trajectory executed by the simple control system in the aforementioned environment is shown in black, as it was in Figure 5.1. The black square represents the goal state.

Using exactly the same wind gust data which was randomly generated for the original execution, the simple control system is applied to execute the manoeuvre sequence used previously. This time, however, the LPM is used to replan midway through the trajectory, using the actual vehicle state at this instant. The replanning instant is chosen exactly half way between the initial and final time, the latter as predicted by the initial execution of the LPM. After the replanning instant, the control system is tasked to execute the new manoeuvre sequence generated by the LPM.

As is clearly visible in Figure 5.3, replanning significantly reduces the final position error in this case, at the cost of an increased energy expenditure. It is important to note that the energy expended is still less than for the path following control system, as seen in Figure 5.1. Through repetition of this experiment in similar environments, we see that this result is repeatable. The exact reduction in error and corresponding increase in

energy expenditure are, however, difficult to quantify, as these parameters are highly dependant on the wind strength, wind direction and position of the goal state relative to the initial vehicle state.

The same experiment is also carried out for two and three replanning events. The results of these experiments are summarised in Table 5.3. In general, more frequent replanning is seen to lead to smaller position errors at the cost of a higher energy expenditure. Energy increases more or less linearly with each additional replanning event, but the corresponding decrease in position error gradually becomes smaller. In other words, the relationship between energy expended and decrease in position error is not a linear one. Replanning is therefore an effective technique by which to reduce position error, but only up to a certain point, beyond which the small additional reduction in position error does not justify the large corresponding increase in energy expenditure.

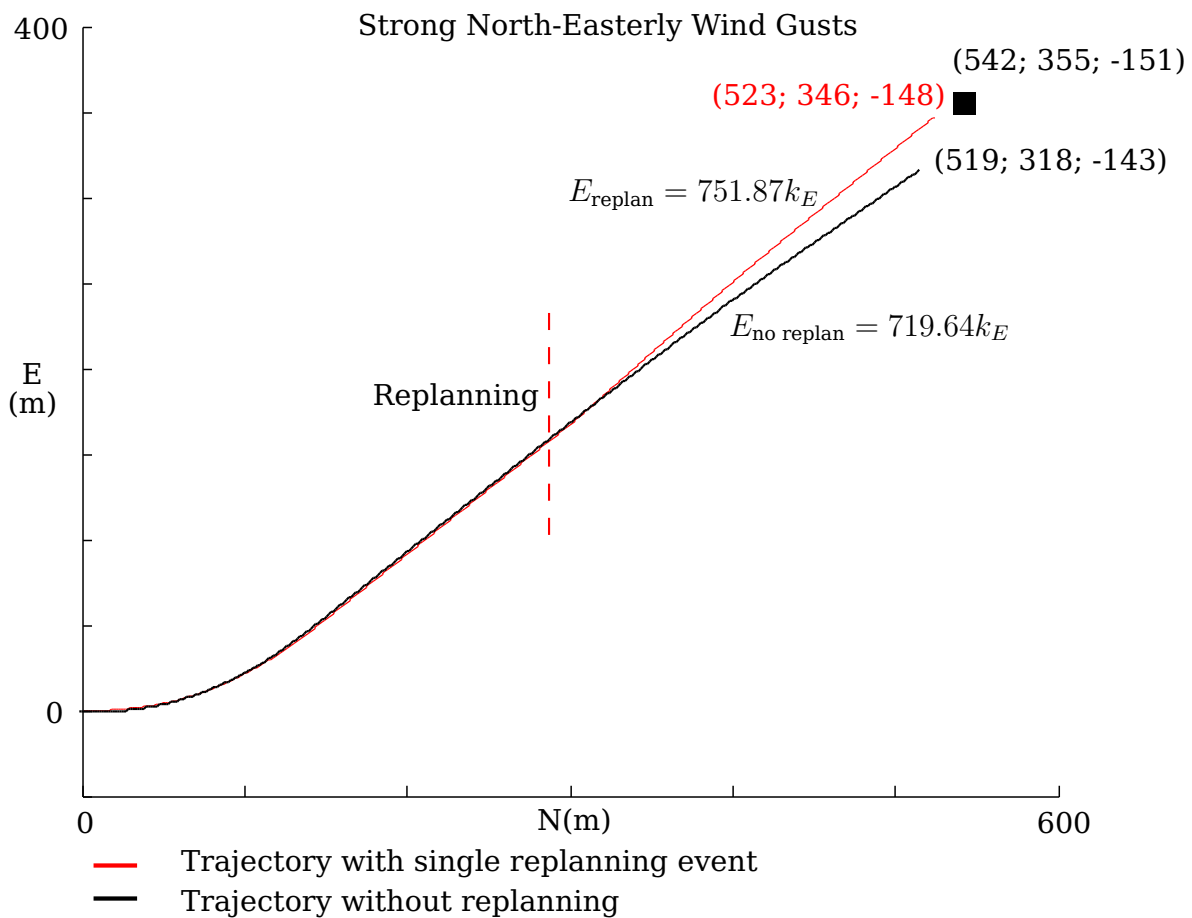


Figure 5.3 – Comparison of trajectories executed by the simple control system (refer to Section 4.2) with and without the use of replanning. The black square represents the goal. The symbol k_E is the proportional constant defined in Equation 5.3.4

5.5 Kinodynamic Planning Algorithm Evaluation

In this section we verify the performance of the PRM (with generic LPM) in various environments which conform to the specifications stated in Section 1.3. That is to say,

Table 5.2 – Effect of different number of replanning events using the LPM in an environment with strong wind gusts. Relative amounts are indicated as percentage offsets from the baseline (0 replanning events).

# Replanning Events	Position Error		Energy	
	Absolute	% Decrease	Absolute	% Increase
0	227.3m	–	719.6 k_E	–
1	165.7m	-27.1%	751.9 k_E	+4.5%
2	140.7m	-38.1%	765.8 k_E	+6.4%
3	130.3m	-42.7%	781.1 k_E	+8.5%

we verify that the PRM-implementation works in environments which are dynamic and cluttered, as well as in environments which require replanning. To this end, we define four test scenarios.

The first scenario is a static maze with walls which extend infinitely upwards, downwards and outward, forming a narrow slalom course which is wide enough for our vehicle to navigate, but not to turn around. This will act as a baseline test in which it is easy to verify visually that milestones are sampled uniformly and that solution trajectories are conflict free. Furthermore, this type of environment is common in many analyses of the PRM. Notable, nearly identical environments appear in several publications by Frazolli [1; 11].

A second scenario consists, initially, of an infinite wall which blocks passage of the vehicle between its initial state and goal state. At a predetermined time, a gap in the wall begins to open gradually, allowing the vehicle passage. This test is presented to demonstrate safety in scenarios where the motion planner must enable the vehicle to bide time safely until a route to the goal becomes unconflicted. This requirement may be important to many real-world scenarios, such as when an aircraft is not allowed to pass through a certain restricted or prohibited air zone until it is cleared to do so by an air traffic controller.

The penultimate test generates environments with large numbers of dynamically-moving obstacles, occupying up to 30% of the entire physical space, with randomly-sampled initial positions and velocity vectors. The kinodynamic planner is tasked to plan routes through many random configurations of this environment. The purpose of this test is to verify that the planning algorithm is reliably able to solve cluttered, dynamic environments.

The final environment tests the ability of the motion planner to address unanticipated changes in the environment through replanning. In this environment, an additional object is introduced directly in the path of the vehicle, after it has already started execution of its originally-planned trajectory. The motion planner is then required to calculate an alternate trajectory in a short period of time.

For each environment, we analyse the distribution of execution times for the motion planning algorithm as well as the distribution of cost for the solution trajectories found. In all cases, we define the cost of a trajectory simply as its path-length.

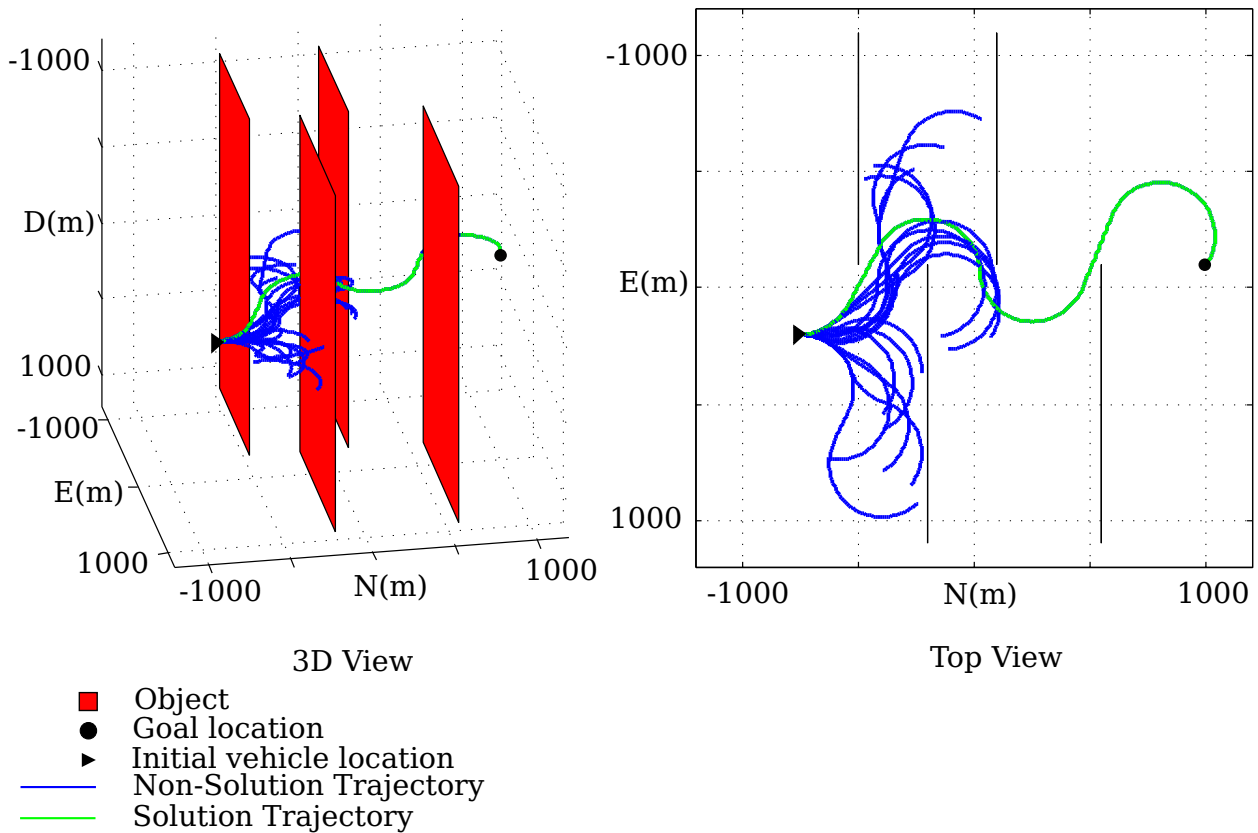


Figure 5.4 – Sample solution in Test Scenario 1: Static Slalom (100 trials).

5.5.1 Test Scenario 1: Static Slalom

The first test scenario, shown in Figure 5.4, is a static slalom. The slalom walls extend infinitely upward, downward and outward – towards the edge of the environment – forming a single, narrow admissible path which is wide enough for our vehicle to navigate, but too narrow for the vehicle to turn by more than about $\pi/2$ radians. We use this as our baseline scenario, since the obstacles are large with clearly defined boundaries, and are stationary, making it easy to visually verify that solution trajectories are valid and conflict-free.

The PRM-based kinodynamic planning algorithm with generic LPM is applied to the problem for one hundred executions. During each execution, the time to first solution is recorded.

This experiment is first performed using a specific goal time associated with the goal state. This time is purposefully chosen to be significantly greater than the minimum time in which the goal state can be reached by the vehicle. This implies that the planning algorithm must extend the sampling space at the outset in order to consider longer paths so that longer trajectories to goal may be generated. We expect that this will, in effect, negate the benefit of dynamically scaling the sampling space as described in Section 3.3.2.

For this test, the algorithm successfully finds a valid, conflict-free solution in all cases.

The distribution of planning algorithm execution times until first solution are presented in Figure 5.5 and are summarised in Table 5.3. As is clear from the data, execution

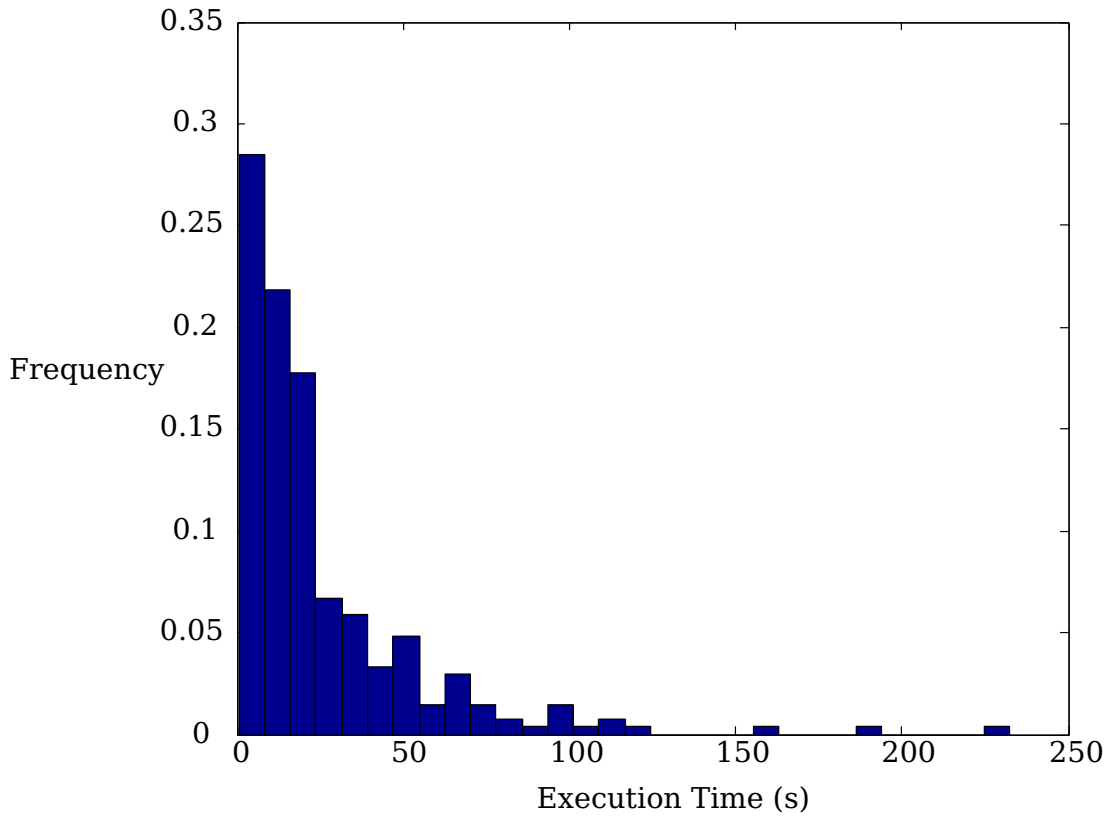


Figure 5.5 – Distribution of execution time until first solution for Test Scenario 1 with specific goal time.

Table 5.3 – Distribution of execution times until first solution for Scenario 1 with specific goal time. The symbol μ denotes the mean, σ denotes standard deviation and $P95$ is the 95% confidence interval

μ	σ	$P95$
26.1s	30.4s	87.1s

times vary from several seconds to well over two minutes. The 95% confidence interval is nearly a minute and a half. That is to say, we expect that the planning algorithm will take at most that long to produce the first solution in the vast majority (95%) of cases. Execution times on the order of minutes are obviously not appropriate for real-time implementation of the algorithm, considering that our vehicle has a nominal velocity of 25ms^{-1} .

We repeat the experiment detailed at the beginning of this section, but no longer require the vehicle to reach its goal at a specific time. This allows the planning algorithm to dynamically scale the sampling space as explained in Section 3.3.2. Additionally, for each trial, the algorithm is allowed to continue executing for a maximum total time of ten seconds regardless of how many solutions it finds.

The cost of the best solution found within this time limit is stored as a differential per-

centage above the optimal solution cost¹.

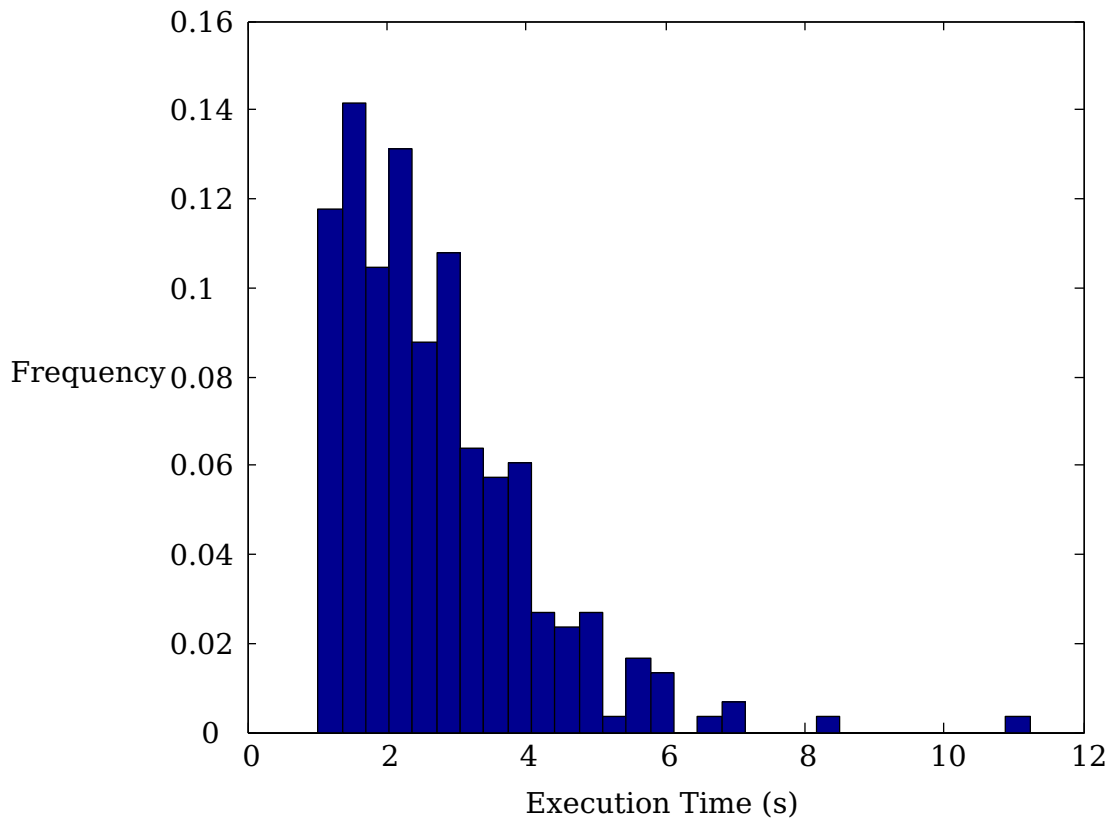


Figure 5.6 – Distribution of execution time until first solution for test scenario 1 with unspecified goal time (100 trials).

Table 5.4 – Distribution of execution times and trajectory costs, both until first solution and after ten seconds, for Scenario 1 with unspecified goal time. Costs are relative to the minimum solution cost for the environment.

First Solution				Best Solution Within 10s	
Execution μ	Execution σ	Cost [†] μ	Cost [†] σ	Cost [†] μ	Cost [†] σ
2.7s	1.4s	+51.4%	38.3%	+15.1%	19.5%

[†] In one trial, no solution was found within ten seconds. This result was disregarded in calculating the mean and variance of trajectory costs.

For this second experiment, distributions of execution time as well as trajectory cost, both for the initial solution and the best solution after ten seconds, are summarised in Table 5.4. The distribution of execution times until first solution are shown in Figure 5.6. We see a marked decrease in both the mean and standard deviation of execution times compared with the first experiment in which the time at the goal state was specified.

¹In this evaluation, we define the optimal solution as the trajectory with the shortest associated path-length. We take the optimal solution as the minimum path-length solution found after applying the PRM for an indefinite period, terminating only after a full hour without any improvement to the solution. This is not necessarily the true optimal solution, but it is considered a sufficiently accurate approximation.

From this we deduce that dynamic scaling of the sampling space significantly reduces execution time of the PRM-algorithm. To verify that this is indeed the reason for the vast improvement in performance, we repeat the first experiment, but specify the goal time to a value which is not significantly longer than the time required by the optimal cost path to the goal. As expected, we observe a dramatic reduction in execution times, compared with the first experiment.

Unfortunately, there is no trivial way to reduce algorithm execution times if we require our vehicle to reach the goal state at a specific instance in time.

At least for an unspecified goal time, execution times of the motion planning algorithm appear promising for real-time implementation. As indicated in Table 5.4, execution times are on the order of seconds. Taking the value of 95% confidence interval as before, we can estimate that, in the vast majority of cases, our planning algorithm will find a solution for the environment under consideration in under 7 seconds. In fact, as seen in Figure 5.6, for one hundred executions, the algorithm only once took more than 10 seconds to find a solution.

In a relatively short period of ten seconds, the motion planning algorithm is shown to find solutions which are generally within 15% of the optimal trajectory cost. The optimal trajectory is determined approximately using the same motion planning algorithm, allowed to execute until it fails to produce a lower cost solution for more than an hour of continuous execution. It is important to note that the mean and variance for the best solution found within ten seconds have been calculated as if the results are distributed normally. As can be deduced from the large variance of best solution cost in Table 5.4, the distribution has a large “tail” towards higher costs. We can therefore say that it is only reasonably likely that the motion planning algorithm will produce a solution with a reasonably low cost for this environment within a short execution time, but in many cases it will initially produce a very expensive solution.

5.5.2 Test Scenario 2: Slowly-Opening Gate

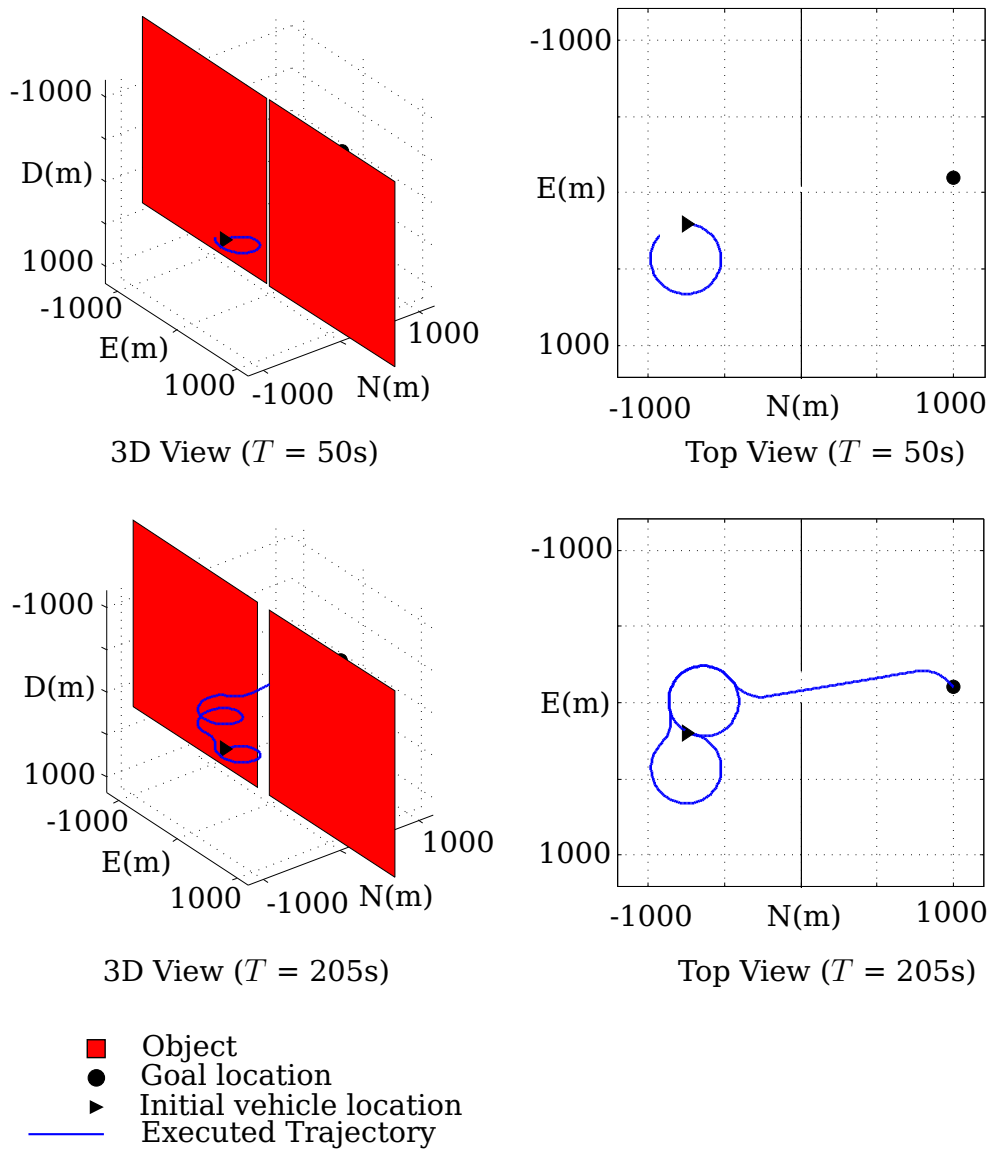


Figure 5.7 – Sample solution in Test Scenario 2: Slowly-Opening Gate.

Having proved that our motion planning algorithm is able to solve a relatively complex cluttered environment, we proceed to scenarios which contain dynamic objects. Scenario 2, shown in Figure 5.7, contains a gate-like obstacle, which initially completely isolates the goal state from the initial vehicle state. At a predetermined time, the gate starts opening slowly, eventually allowing passage to the vehicle.

The purpose of this scenario is to verify that the motion planning algorithm is able to cope safely with situations in which obstructions or otherwise impassable areas completely cut off all routes between the vehicle and its goal state. A typical example in which such a scenario may arise for a UAV, is while waiting for clearance into a restricted air zone in or beyond which lies its goal. The desired solution in this scenario is a trajectory which causes the vehicle to bide time safely until conditions in the environment are such that the goal state is reachable, and subsequently continues to the goal state.

As in the previous section, we carry out two experiments, in each of which we apply the planning algorithm one hundred times: first with an associated time at the goal state, and subsequently with the time unspecified.

Table 5.5 – Distribution of execution times until first solution for Scenario 2 with specific goal time (100 trials).

μ	σ	$P95$
55.5s	77.9s	211.3s

As before, the execution times in the first experiment, summarised in Table 5.5, are on the order minutes, with many solutions taking up to three minutes or more. This result further verifies our earlier conclusion that using our motion planning algorithm to find trajectories which must reach the goal state at a specific time, is not viable for real-time implementation on current-generation microprocessors.

Table 5.6 – Distribution of execution times and trajectory costs, both until first solution and after 30 seconds, for Scenario 2 with unspecified goal time (100 trials). Costs are relative to the minimum solution cost for the environment.

First Solution				Best Solution Within 30s	
Execution μ	Execution σ	Cost μ	Cost σ	Cost μ	Cost σ
12.3s	11.4s	+19.5%	24.8%	+15.5%	17.9%

We again observe a marked reduction in execution times when no goal time is specified, as indicated by the results in Table 5.6. Using the 95% confidence we can estimate that most executions of the motion planner complete in slightly over half a minute. As we consider this to be a relatively long execution time – likely too long for real-time implementation – we limit the motion planner to thirty seconds for one hundred trials to find the solution cost distribution summarised in Table 5.6. Remarkably, all trials complete in under half a minute. As can be expected from the fact that many of the trials require nearly as long to find an initial solution, we do not see a marked difference between the cost of the first solution and the cost after thirty seconds.

The initial path cost is, however, lower relative to that of the optimal trajectory than was the case in Scenario 1. This can be attributed to the fact that the gate in Scenario 2 opens very slowly, leaving only a small gap and thus a relatively narrow range of possible solutions compared with Scenario 1.

In summary, the motion planning algorithm is relatively slow to find an initial solution in this environment. Execution times, routinely bordering on half a minute, are probably too slow for real-time implementation. The initial trajectories found are, however, of a relatively low cost compared to the optimal solution. Bearing in mind that much of the code is implemented in Matlab, reductions in execution time through simply porting the code to the C language could potentially reduce these times sufficiently for this purpose.

5.5.3 Test Scenario 3: Cluttered Dynamic Environment

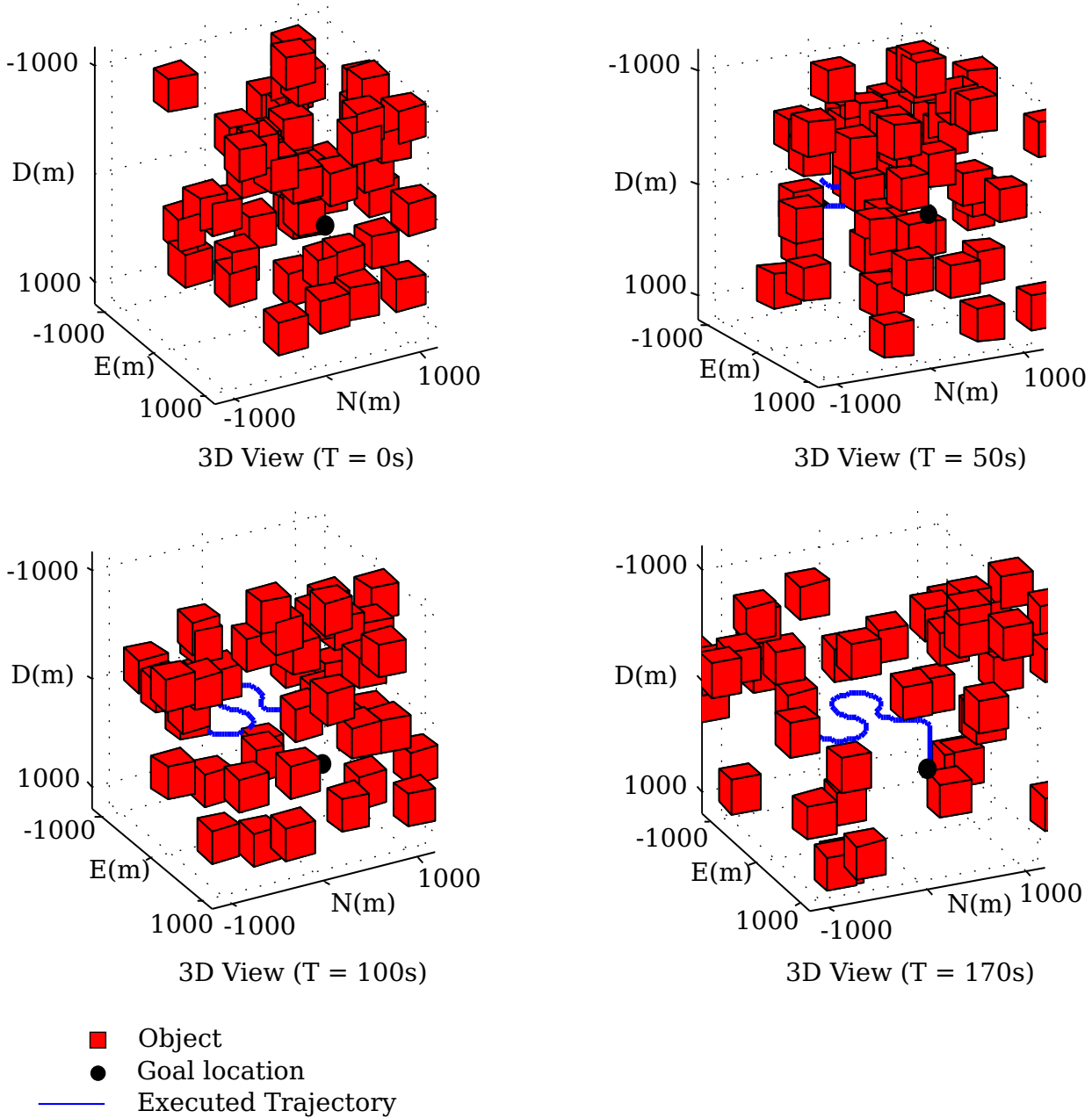


Figure 5.8 – Sample solution in Test Scenario 3: A Cluttered Dynamic Environment.

It has been verified that the motion planning algorithm developed is capable of handling simple static and dynamic environments. An additional requirement which requires testing, is the ability to generate safe trajectories through cluttered environments. We also wish to gauge the influence of the degree of clutter in the environment on planning algorithm execution times.

In order to cover as broad a spectrum of cluttered environments as possible, we develop a scenario which is randomly populated with cubes of equal size. An example of this environment is shown in Figure 5.8. Each cube has a randomised initial position and

a randomly generated velocity vector of a magnitude between zero and the maximum speed of the vehicle. A random number of cubes, within a range which will lead to 0% to 40% clutter, are generated for every instance of the environment. That is to say, the volume of space occupied by the cubes ranges between 0% and 40% of the total environment volume. Cubes are not allowed to have an initial position which lies within a certain radius from the initial or the goal state, as this may cause immediate, unavoidable conflict in the former case, or an inability to reach the goal state due to a stationary cube which encloses it, in the latter. Cubes may intersect and thus transiently form larger compound objects.

Based on results from the previous two scenarios which indicate that the motion planning algorithm developed is not suitable for planning a trajectory to the goal state with a specific associated time, we only analyse the case in which no specific time is associated with the goal.

We define the clutter factor γ_c as the ratio of space which is occupied relative to the total volume of the environment or

$$\gamma_c = \frac{V_{\text{occupied}}}{V_{\text{total}}}. \quad (5.5.1)$$

As objects in this scenario are allowed to intersect, calculating γ_c is not a trivial matter. We can, however, determine an estimate of this value if we know the number of objects in the environment and their volumes (note that the objects are identical). As shown in Appendix B, we can approximate the clutter factor by

$$\gamma_c \simeq 1 - \frac{(C - q)^n}{C^n}, \quad (5.5.2)$$

where

C is total volume of the environment,

q is the volume of one of the objects which occupy the environment, and

n is the number of objects in the environment.

This approximation disregards variance in the volume of intersection of objects and the change of intersection over time. Furthermore, it relies on the assumption that $C \gg q$. It is therefore only a rough estimate, but it should be sufficient for our purposes.

Once again the motion planning algorithm is applied one hundred times, for several discrete clutter factors between $\gamma_c = 0.0$ and $\gamma_c = 0.4$. The scenario is randomly generated for each trial, meaning that the results obtained are indicative of algorithm performance in a general cluttered environment which is 0% to 40% occupied, rather than being applicable to one specific environment configuration.

The distribution of execution times is presented in Figure 5.9. As the clutter factor approaches $\gamma_c = 0.4$, execution times are found to be on the order of several minutes and for clutter higher than $\gamma_c = 0.7$, the algorithm is found to execute for over an hour without a solution in several cases. This dramatic increase in execution time suggests some exponential relationship between the clutter factor and execution time for our motion planning algorithm. Exponential-curve fits for the mean, standard deviation and 95% confidence interval are shown in Figure 5.9 as dashed lines.

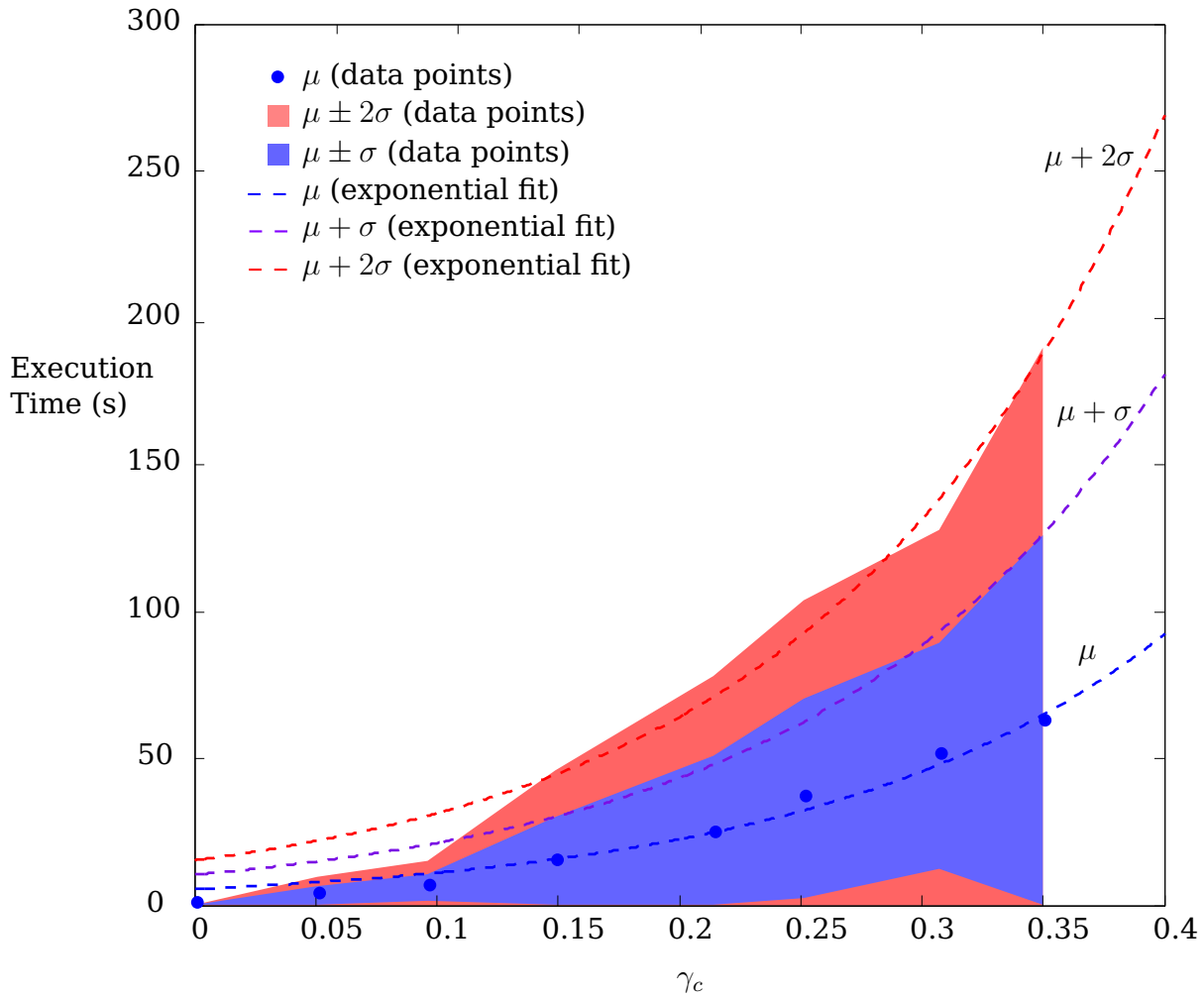


Figure 5.9 – Relationship between environment clutter factor and PRM-algorithm execution times. The discrepancy between actual data points and the exponential curve fit for low clutter factors (lower than about $\gamma_c = 0.1$) can be attributed to the dominant role of the LPM in environments with little clutter.

The exponential curves are found to fit the collected data points very well, except for clutter factors lower than about $\gamma_c = 0.1$. This can be attributed to the dominant role of the LPM in relatively empty environments: when little or no conflict arises, few milestones need to be generated before a solution is found and a direct solution can be obtained mostly using the LPM.

From Figure 5.9, it is evident that, in addition to an exponential growth of the mean execution time, the standard deviation also increases exponentially with increased clutter factor. This means that, in highly cluttered environments, it does not only take longer to find solutions, but it is also more difficult to predict how long the algorithm will execute before the first solution is found.

For environments with about 5% clutter, the motion planning algorithm is seen to complete in under ten seconds in more than 95% of all cases, with a mean execution time of just over three seconds.

It is important to note that the results summarised here are averaged for many different configurations of the environment. Even for extremely high clutter factors, it is always possible to arrange objects in the environment such that free space in the environment is well-connected and the motion planning problem therefore becomes trivial. For example, an environment may be 99% occupied (i.e. $\gamma = 0.99$), but the objects arranged in such a manner that they form a perfectly straight and static corridor around the optimal path between the initial and goal states. For such a well-connected environment, the motion planning algorithm may find the optimal solution immediately by simply applying the LPM once. For random permutations of a highly-cluttered environment, such configurations are, however, extremely unlikely: randomly distributed clutter usually leads to a badly-connected environment.

Now that we know the relationship between clutter and the execution time of our motion planning algorithm, we may draw some conclusion as to the safety of these execution times with regard to real-time implementation. The first obvious factor is what range of clutter factors may be common in real-world environments.

Let us consider a worst-case scenario: an extremely busy (i.e. very cluttered) real-world airspace. Los Angeles International Airport (LAX) has one of the ten busiest airspaces in the world with over six hundred thousand aircraft operations annually [53]. Measurements in the mid-1980's indicated peak aircraft densities of up to 0.10 aircraft per cubic kilometre² [54]. Although air traffic has not increased substantially in the Los Angeles area in decades [55], we shall use a density of 0.20 (twice the measured peak density) for safety reasons.

The United States Federal Aviation Administration distinguishes between three types of near-collision conflict encounters [56]:

- Critical encounter: aircraft separation is less than 100ft (less than about 31m).
- Potential encounter: aircraft separation is between 100ft and 500ft (between about 31m to 152m).
- No hazard: any separation greater than 500ft (about 152m).

An exclusion radius of 500m for each aircraft, which is safe by the aforementioned standard, and an aircraft density of 0.20 per cubic kilometre, as stated above, leads to a clutter factor of only about $\gamma_c = 0.10$. Even increasing the aircraft density four times above measured peaks only gives $\gamma_c = 0.21$. Note that we do not use Equation 5.5.2 to find the clutter factor in this case, since exclusion zones are not allowed to intersect. In this case we may apply Equation 5.5.1 directly.

Bearing in mind that the assumed aircraft density (and therefore also the clutter factor) is significantly higher than empirical data suggests is typical, we can make the assumption that during normal flight³, an aircraft would not encounter environments with a clutter factor greater than about $\gamma_c = 0.2$, and will generally operate in environments with clutter factors lower than $\gamma_c = 0.1$.

²In the published results [54], the aircraft density is expressed as number of aircraft per *square* mile. This is, however, equivalent to the number of aircraft per cubic mile, considering that all aircraft were detected by measuring equipment, regardless of altitude.

³By "normal flight", we mean under conditions where the aircraft is not required to operate close to terrain, such as is the case during takeoff and landing.

From Figure 5.9, we can see that the motion planning algorithm finds solutions in such environments within seconds. This indicates that the planning algorithm is probably fast enough for real-time application in most real-world environments. For situations where an aircraft is required to operate in an environment with unusually high clutter, for example a UAV performing low-altitude surveillance in an urban environment, the execution times of the planning algorithm become disturbingly high. For such scenarios, further investigation of the properties of each specific operating environment is warranted, as the algorithm will not necessarily be able to provide solutions in real-time for environments where the free space is very badly connected (γ_c larger than about 0.2).

5.5.4 Test Scenario 4: Dynamic Environment With Replanning

We have shown in Section 5.4.2 that replanning may reduce the effect of a disturbance such as wind, but our planning algorithm should also be able to cope with uncertainty in the environment through a periodic replanning approach. Uncertainty may include, among other factors, uncertainty in the position and velocity of the vehicle and of objects in the environment, both current and propagated into the future. Up until this point, we have assumed accurate and exact knowledge of the external environment and of the vehicle state, but in practice, this is usually far from the case. In practice, the motion planner must rely on uncertain and often incomplete information from the modelling module.

To test the ability of the motion planning algorithm to cope with unexpected changes in the environment, we introduce a simple scenario with three objects, only two of which are initially “visible” to the the planning algorithm. We allow the algorithm to plan a route to the goal state for ten seconds and then let the vehicle begin executing the planned trajectory in simulation. At some point during execution of the trajectory, we place a third object directly on the nominal planned path ahead of the vehicle and require the algorithm to replan the rest of the trajectory with knowledge of all three objects. In practice, such a replanning event might be triggered on a time-based schedule, say every five seconds, or every time a significant change in the environment map occurs in the modelling module. The sudden and unexpected “appearance” of a new object may be due to it coming into the range of the on-board sensors for the first time, or due to a previously detected object which followed a different trajectory to that initially predicted by the modelling module.

We apply the motion planning algorithm to this scenario one hundred times. An example execution is presented in Figure 5.10, showing the initially planned trajectory, the instant at which replanning occurs and the actual executed trajectory at the end of the mission.

The algorithm execution time distribution is summarised in Table 5.7.

Table 5.7 – Distribution of execution times and trajectory costs for Scenario 4 replanning stage. Costs are relative to the minimum solution cost for the environment.

First Solution				Best Solution Within 10s	
Execution μ	Execution σ	Cost μ	Cost σ	Cost μ	Cost σ
0.33s	0.32s	+14.8%	20.3%	+0.0%	0.0%

For such a simple environment, replanning is found to be rapid, providing low cost solutions in the order of milliseconds and optimal trajectories within seconds. It must be noted that, as with all applications of the motion planning algorithm, the efficiency of replanning is highly dependant on the clutter factor of the environment.

From our results, it can be concluded that replanning is an effective approach for coping with uncertainties in the environment, given that the clutter factor of the environment is sufficiently low to allow real-time planning from the outset, and that this clutter factor does not change significantly over time.

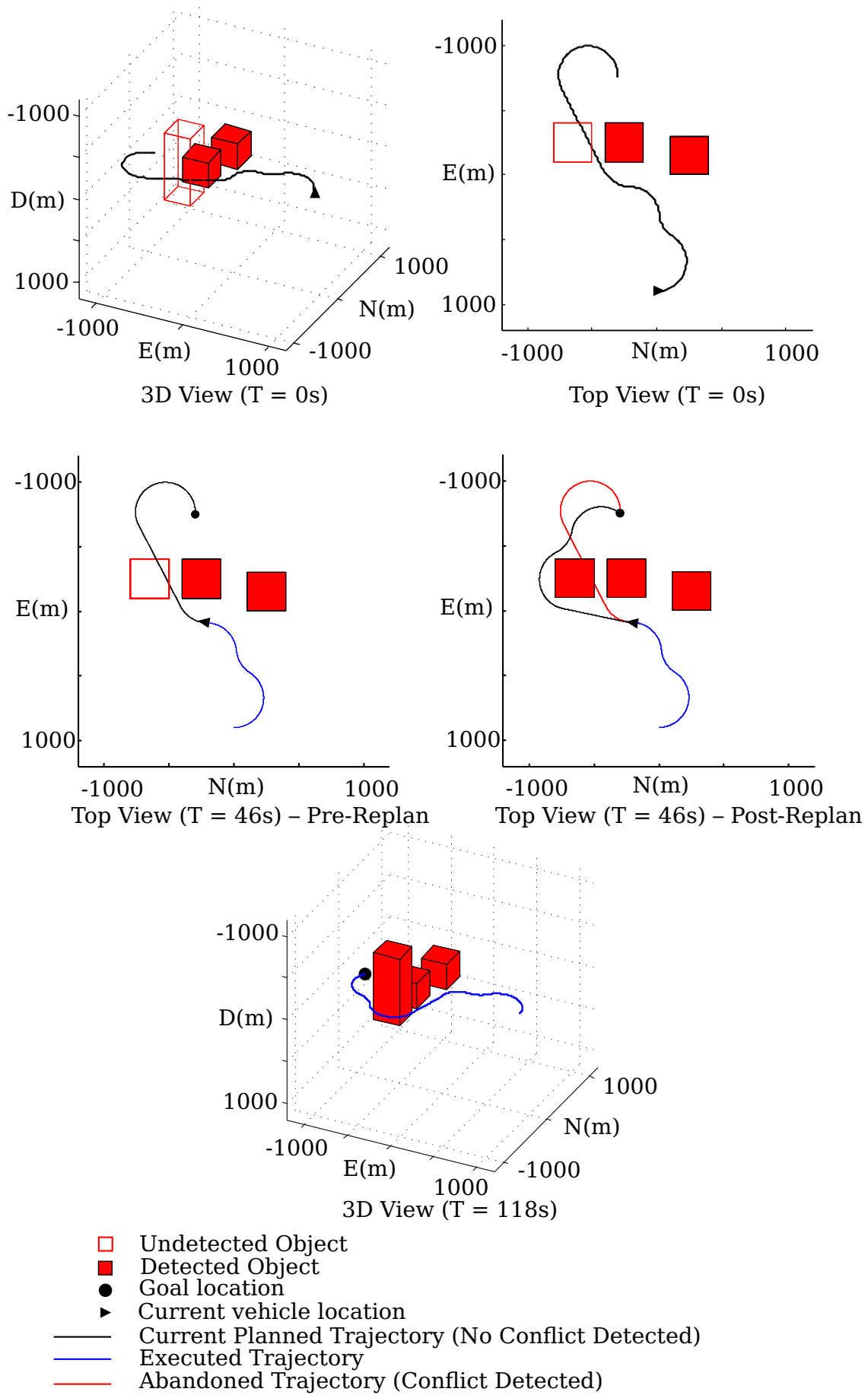


Figure 5.10 – Sample solution in Test Scenario 4: Dynamic Environment with Replanning.

5.5.5 Per-Function Breakdown of PRM-Algorithm Execution Times

In order to improve the PRM-algorithm, it is useful to know which parts of the code act as bottlenecks to its performance. To this end, we use a profiling tool to analyse the code as the PRM executes in the various environments presented in the previous sections. A summary of the functions which were responsible for using most of the processing time, is presented in Table 5.8 below.

Table 5.8 – Per-Function breakdown of time spent in PRM-Algorithm.

Function	% Time in Function
Conflict Detection	64.9
LPM (Determine Manoeuvre Trajectory)	25.9
Graph Data Structure Operations	7.0
LPM (Other Operations)	1.5
Sampling Milestones	0.05
Various Other Functions	0.65

The list of processor-intensive items is topped by the mock conflict detection function (see Appendix A). This is cause for concern. Since a practical conflict detection module must process real environment data obtained using real, limited-range sensors with noise and uncertainty, as opposed to explicitly-defined objects in a simulated environment, we expect a real-world implementation of the conflict detection module to be even slower – possibly considerably slower – than our function, which already accounts for nearly two thirds of the total execution time.

Another item high on the list is the function which determines nominal manoeuvre trajectories. This is not surprising, considering that manoeuvres are generated on the order of hundreds of times for each call to the LPM. The time required by the LPM can possibly be reduced by optimising the memory accesses done when copying manoeuvre data from the hash-map-like structure which stores them. As the manoeuvre data for our UAV is relatively small (only 8MB uncompressed), most of this data could possibly be forced into fast cache memory. The effect of doing so is difficult to predict, since many modern compilers make such optimisations on their own accord. It is, however, definitely an avenue which warrants investigation.

Operations on the PRM graph rank high in Table 5.8, but this may be deceiving, as nearly half of the total time spent on graph operations consist of overhead in converting C-arrays to Matlab matrices using the *libpointer* interface provided by Matlab. If both the graph structure and PRM were to be implemented in C, this overhead could be eliminated.

Judging from the fact that milestone sampling, including sample rejection based on a subset of vehicle constraints, takes up only a fraction of the total execution time, we confirm our assumption that this process is very efficient.

Chapter 6

Conclusion

6.1 Summary

This thesis describes the development of a generic Local Planning Method (LPM) which is able to use implicitly-defined motion primitives or vehicle *manoeuvres* to construct the optimal nominal trajectory between two state-time pairs in an obstacle-free space. This LPM is integrated in a larger motion planner, based on the Probabilistic Roadmap Method (PRM) algorithm, which is shown to be applicable to dynamic, cluttered, three-dimensional environments. The viability of this motion planner is demonstrated for the case of a small Unmanned Aerial Vehicle using software simulations of a vehicle model which was previously verified using practical flight tests. A brief summary of each chapter follows.

Chapter 1

Background to the motion planning problem, particularly in the context of vehicles with non-holonomic constraints, is given. The architecture for an integrated autonomous navigation system is presented and research objectives for the project are defined. Finally, the test UAV, of which the model was used to test the algorithms developed in this paper, is introduced.

Chapter 2

A literature study of general concepts used throughout the thesis, including axes definitions and the model of a fixed-wing UAV and notations used to express system dynamics, is provided. The specific control system architecture used to implement manoeuvres for our test vehicle is also introduced.

Chapter 3

Chapter 3 is divided into two sections. In the first section, a literature study of existing motion planning algorithms is presented. One of these algorithms, the Probabilistic Roadmap Method (PRM), is chosen for this project. The second part of the chapter provides a high-level overview of the motion planning algorithm chosen, as well as implementation details. Some modifications to the original PRM-formulation are proposed.

Chapter 4

This chapter is dedicated to a specific constituent component of the PRM: the Local Planning Method (LPM). The classic LPM-implementation, based on simple geometrically-defined motion primitives, is discussed at length. The negative aspects associated with the use of geometrically-perfect manoeuvres are explained, and an LPM which uses

implicitly-defined manoeuvres which are more natural for the test vehicle is proposed. A simplified control architecture to effect such manoeuvres is also proposed. The LPM is formulated as a minimisation problem, and a specific minimisation algorithm is chosen for its implementation, based on a comprehensive literature study of common minimisation techniques. The detailed design of the LPM is discussed.

Chapter 5

The penultimate chapter details several experiments which verify the LPM, and the larger motion planning algorithm (the PRM) which incorporates it. Several test environments are chosen, each of which verifies different aspects or requirements of the planning algorithm.

6.2 Primary Contributions and Notable Results

The following is a list of contributions and notable results.

1. It is shown that Dubins-like helices and straight line motion primitives are fundamentally inappropriate approximations of energy-efficient manoeuvres for a fixed-wing UAV. Furthermore, it is demonstrated that the aggressive control techniques typically employed to track position trajectories which are based on such geometrically perfect primitives, are quite energy intensive for real-world applications. It is shown that aggressive position tracking consumes significant energy. This is considered wasteful, since the uncertainty in the future states of other entities in the environment is usually very high, meaning that there is little benefit in reducing the position-tracking error of the vehicle beyond a certain threshold.
2. A less aggressive control system, which does not regulate specific axial acceleration, is developed. A means is devised to define and represent implicit vehicle manoeuvres based on the nominal trajectories produced by this control system for a specific vehicle. In moderate winds, the new, simpler control system is shown to consume up to 8% less energy compared with a path-following control system, at the expense of accuracy in the final position. It is shown that, through periodic replanning using the LPM, drift from the nominal position can be reduced to an acceptable level, if so desired.
3. The most notable contribution of this project, is the development of a novel, generic Local Planning Method (LPM), which is able to connect two state-time pairs optimally using the implicitly-defined, discretised manoeuvre trajectories referred to in Item 2 above.
4. The LPM referred to in Item 3 is integrated in a larger motion planning algorithm based on the Probabilistic Roadmap Method (PRM). This motion planner is shown to provide solutions in dynamic, cluttered environments. It is also demonstrated that the motion planner is able to compensate for unexpected changes or uncertainty in the environment through periodic re-planning.
5. Execution times for the new LPM are found to be of the order of milliseconds on a typical desktop CPU, which is relatively long compared with classical LPMs which are able to find the optimal connection between two state-time pairs analytically in constant time. It is, however, also demonstrated that the PRM utilising this LPM

is able to find solutions rapidly in dynamic environments with a degree of clutter equivalent to several times that which is typically encountered by commercial aircraft. In more complex environments, with an object-to-free space ratio greater than about 20%, the execution times of the PRM-algorithm unfortunately become too high for feasible real-time implementation.

6.3 Future Work

There are many avenues of investigation which may lead to improved performance of either the Local Planning Method or the larger motion planning algorithm which uses it. We highlight some of the more interesting possibilities:

1. The PRM-algorithm generates milestone state-time pairs in batch. It subsequently applies the LPM to each milestone in the batch, paired with each node in a graph of states known to be reachable. This often leads to cases where the PRM generates a long queue of state-time pairs and then waits while they are connected by the LPM. Currently this is done sequentially, in a single thread. Modern CPUs with multiple cores present the possibility of running multiple LPM threads simultaneously. Since calls to the LPM currently account for more than a quarter of the entire PRM execution time, multi-threading the LPM may lead to significant improvements in the overall performance of the motion planner.
2. A new, asymptotically optimal extension to the PRM, known as PRM* [25] was published after the inception of this project. The PRM* is completely compatible with the original PRM, but provides the added benefit that solutions provably converge to the optimal trajectory. An implementation of the PRM*-algorithm which uses the LPM presented in this project, should be able to provide better solutions in cases where the algorithm has ample execution time.
3. The FLMA minimisation algorithm which is used in the LPM, could potentially be applied to optimise solutions found by the PRM very rapidly, similar to the way in which the LPM is currently applied to optimise the path-lengths of the constituent primitives in a manoeuvre sequence. Metaphorically, the original path produced by the PRM may be visualised as a slack string which connects the initial and goal states with unnecessary twists and turns. The FLMA minimisation can then be used to “pull the ends of the string until it is taut” (i.e. adjust the length of manoeuvres along the trajectory until the total path length reaches the global minimum).
4. Neither the conflict detection model, nor the modelling module was explored in detail in this project. Once these modules are implemented, the algorithms developed in this project may be tested in conjunction with these modules to give a better indication of the feasibility of the entire proposed autonomous navigation system.
5. This project assumes the lack of coordination of conflict avoidance between different entities in the environment. If some automated traffic coordination system exists which is able to assist multiple vehicles in the environment in planning trajectories which do not intersect, the safety of all vehicles in the environment may be greatly increased. This system need not be centralised: it could be possible to devise a distributed protocol which allows vehicles to relay their intentions to one another and to reach a mutual agreement on a safe trajectory for each vehicle.

Appendix A

Simulating the Environment and Conflict Detection

The simulated environments which are used to test the motion planning algorithm developed in this project, are constructed from *patch* objects in Matlab. These objects are three-dimensional polygonal surfaces which are defined by a matrix of vertices. We define a *getObjects(simulationTime)* function which returns a list of all objects in the environment. Simulation time is provided as an input parameter to this function, since objects are dynamic and therefore have different positions at different time instances.

A global conflict detection function is provided, which has access to the aforementioned *getObjects* function. The conflict detection function takes a time-series vector and position-based trajectory matrix as input arguments.

```

1: function ConflictDetect(t, P)           ▷ Time vector and position matrix as arguments
2:   for each point  $p$  in  $P$  do
3:     simtime  $\leftarrow$  time at  $p$  reached
4:     for each object  $O$  in getObjects(simtime) do
5:       for each surface  $s$  of  $O$  do
6:         if distance( $s, p$ ) < threshold then return conflicted
7:         end if
8:       end for
9:     end for
10:  end for
11:  return unconflicted
12: end function

```

As seen in the pseudocode for the conflict detection function above, each position in the trajectory to be evaluated is tested for proximity to every surface of every object at the appropriate simulation time. This is quite an expensive operation, but not nearly as expensive as a call to a real conflict detection module is expected to be.

Note that this function will return false “unconflicted” results if the resolution of the input trajectories are not high enough. An appropriate resolution is determined based on the maximum velocity of an object and of the simulated vehicle.

Appendix B

Estimating the Volume Occupied by a Number of Identical, Uniformly-Distributed and Possibly Overlapping Objects

Consider a large finite space of volume C , containing n small objects of equal volume q each. The objects are distributed uniformly and independently in the space and are not allowed to extend beyond its borders. The objects may intersect. We wish to approximate the expected volume of C which is occupied by one or more of the objects.

The probability that a point in C is occupied by a specific object is given by

$$P_{\text{object}} = \frac{q}{C}. \quad (\text{B.0.1})$$

The probability of the point not being occupied by that same object is then

$$\bar{P}_{\text{object}} = 1 - \frac{q}{C}. \quad (\text{B.0.2})$$

The probability of a point in the space being unoccupied by any of the n independent objects is thus

$$\bar{P}_{\text{occupied}} = \left(\frac{C - q}{C} \right)^n, \quad (\text{B.0.3})$$

and it therefore follows that the probability that a point is occupied by one or more of the objects is

$$P_{\text{occupied}} = 1 - \left(\frac{C - q}{C} \right)^n. \quad (\text{B.0.4})$$

Through integrating Equation B.0.4, we may obtain an estimate of the total volume occupied V_{occupied} by the n small objects,

$$V_{\text{occupied}} = \int_{x \in C} 1 - \left(\frac{C - q}{C} \right)^n dx \quad (\text{B.0.5})$$

We can not directly evaluate Equation B.0.5 to an exact closed-form solution, but under the condition that $C \gg q$, it is possible to approximate the integral as

$$V_{\text{occupied}} \simeq C - \frac{(C - q)^n}{C^{n-1}}. \quad (\text{B.0.6})$$

Using Equation B.0.6, we obtain an estimate for the ratio of occupied volume to total region volume, or *clutter factor*,

$$\gamma_c \simeq 1 - \frac{(C - q)^n}{C^n}. \quad (\text{B.0.7})$$

Appendix C

Pseudocode for Fletcher's Version of the LMA

Algorithm 3 Pseudocode outlining Fletcher's version of LMA

```

1: function FLMA( $s_0, s_g, t$ )           ▷  $t$  passed as parameter is a vector of initial estimates
2:
3:   initialise damping factor  $\lambda$ , bounding value variables and maximum number of
   iterations allowed
4:
5:    $r \leftarrow residuals(s_0, s_g, t)$            ▷ Initial calculations
6:    $J \leftarrow Jacobian(s_0, s_g, r)$ 
7:    $A \leftarrow J^T J$ 
8:    $v \leftarrow J^T r$ 
9:    $S_s \leftarrow r^T r$ 
10:
11:   $D \leftarrow diagonal(A)$            ▷ Scaling matrix
12:
13:  while (incremental changes large) and (iterations  $\leq$  allowed) do
14:    ▷ Begin Outer Iteration
15:     $D_n \leftarrow diagonal(A)$ 
16:    while (incremental changes large) and (iterations  $\leq$  allowed) do
17:      ▷ Begin Inner Iteration
18:
19:      while 1 do
20:         $A_U \leftarrow upper\ triangular\ matrix(A)$ 
21:         $A \leftarrow A_U^T + A_U + diagonal(D_n + \lambda D)$ 
22:
23:        if  $A$  is positive definite then
24:          Solve for  $U$  given  $U^T U = A$            ▷ Use Cholesky factorisation
25:        else
26:          scale  $\lambda$  (increase)
27:          break
28:        end if
29:

```

```

30:         if  $\lambda = 0$  then
31:              $\lambda \leftarrow 1$ 
32:         end if
33:     end while
34:
35:      $\delta \mathbf{t} \leftarrow \mathbf{U} \setminus (\mathbf{U}^T \setminus \mathbf{v})$  ▷ Trial
36:      $\mathbf{t}_{new} \leftarrow \mathbf{t} - \delta \mathbf{t}$ 
37:      $\mathbf{r} \leftarrow \text{residuals}(\mathbf{s}_0, \mathbf{s}_g, \boldsymbol{\tau}_{new})$ 
38:      $S_{s_{new}} = \mathbf{r}^T \mathbf{r}$ 
39:      $\delta S = S_s - S_{s_{new}}$ 
40:
41:     if  $\delta S$  very large then break
42:     end if
43:
44:      $\mathbf{A} \leftarrow \mathbf{U}$ 
45:
46:     if  $\lambda = 0$  then
47:         Recalculate  $\mathbf{A}$  and  $\lambda$ 
48:     end if
49:     scale  $\lambda$  (increase)
50: end while ▷ End Inner Iteration
51:
52:     if  $\delta S$  very large then
53:         scale  $\lambda$  (decrease)
54:     end if
55:      $S_s = S_{s_{new}}$  ▷ Update estimate
56:      $\boldsymbol{\tau} = \boldsymbol{\tau}_{new}$ 
57:      $\mathbf{J} \leftarrow \text{Jacobian}(\mathbf{s}_0, \mathbf{s}_g, \mathbf{r})$ 
58:      $\mathbf{A} \leftarrow \mathbf{J}^T \mathbf{J}$ 
59:      $\mathbf{v} \leftarrow \mathbf{J}^T \mathbf{r}$ 
60: end while ▷ End Outer Iteration
61: return  $\mathbf{t}_{new}$ 
62: end function

```

Bibliography

- [1] Frazzoli, E., Dahleh, M.A. and Feron, E.: Real-time motion planning for agile autonomous vehicles. In: *Proceedings of the American Control Conference*, pp. 43–49. 2000.
- [2] Van Daalen, C.E.: *Conflict Detection and Resolution for Autonomous Vehicles*. Ph.D. dissertation, Stellenbosch University, 2010.
- [3] Smit, S.J.A.: *Untitled draft copy to be submitted*. Master's thesis, Stellenbosch University, 2013.
- [4] Hough, W.J.: *Autonomous Aerobatic Flight of a Fixed Wing Unmanned Aerial Vehicle*. Master's thesis, Stellenbosch University, 2007.
- [5] Gaum, D.R.: *Aggressive Flight Control Techniques for a Fixed-Wing Unmanned Aerial Vehicle*. Master's thesis, Stellenbosch University, 2009.
- [6] De Jager, A.: Photograph: "Sampioen". Electronic Systems Laboratory, University of Stellenbosch, 2011.
- [7] Peddle, I.K.: *Acceleration Based Manoeuvre Flight Control System for Unmanned Aerial Vehicles*. Ph.D. dissertation, Stellenbosch University, 2008.
- [8] Peddle, I.K.: *Autonomous Flight of a Model Aircraft*. Master's thesis, Stellenbosch University, 2005.
- [9] Blaauw, D.: *Flight Control System for a Variable Stability Blended-Wing-Body*. Master's thesis, Stellenbosch University, 2008.
- [10] Etkin, B. and Reid, L.: *Dynamics of Flight, Stability and Control*. 3rd edn. Wiley and Sons, 1996.
- [11] Frazzoli, E., Dahleh, M. and Feron, E.: Real-time motion planning for agile autonomous vehicles. *Journal of Guidance, Control and Dynamics*, vol. 25, pp. 116–129, 2002.
- [12] Bellman, R.E.: *Dynamic Programming*. Princeton University Press, 1957.
- [13] Bellman, R.E.: *Dynamic Programming (Republished Edition)*. Courier Dover Publications, 2003.
- [14] Hsu, D., Kindel, R., Latombe, J. and Rock, S.: Randomized kinodynamic motion planning with moving obstacles. *International Journal of Robotics Research*, vol. 21(3), pp. 233–255, 2002.

- [15] Canny, J.F.: *The Complexity of Robot Motion Planning*. The MIT Press, 1987.
- [16] LaValle, S.M.: From dynamic programming to RRTs: Algorithmic design of feasible trajectories. In: Bicchi, A., Prattichizzo, D. and Christensen, H. (eds.), *Control Problems in Robotics*, vol. 4 of *Springer Tracts in Advanced Robotics*, pp. 19–37. Springer Berlin/Heidelberg, 2003.
- [17] LaValle, S.M., Branicky, M.S. and Lindemann, S.R.: On the relationship between classical grid search and probabilistic roadmaps. *International Journal of Robotics Research*, vol. 23, 2004.
- [18] Barraquand, J. and Latombe, J.: A distributed representation approach. *The International Journal of Robotics Research*, vol. 10, pp. 628–649, 1991.
- [19] Barraquand, J. and Latombe, J.: Controllability of mobile robots with kinematic constraints. Tech. Rep. STAN-CS-90-1317, Department of Computer Science, Stanford University, 1990.
- [20] Švestka, P. and Overmars, M.H.: *Robot Motion Planning and Control*, chap. 5. Lecture Notes in Control and Information Sciences. Springer, 1998.
- [21] Hsu, D., Latombe, J. and Motwani, R.: Path planning in expansive configuration spaces. In: *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 2719–2726. 1997.
- [22] LaValle, S.M.: *Planning Algorithms*. Cambridge University Press, 2006.
- [23] Kavraki, L.E., Švestka, P., Latombe, J. and Overmars, M.H.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, vol. 12(4), pp. 566–580, 1996.
- [24] LaValle, S.M.: Rapidly-exploring random trees: a new tool for path planning. Tech. Rep., Computer Science Department, Iowa State University, 1998.
- [25] Karaman, S. and Frazzoli, E.: Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, vol. 30(7), pp. 846–894, 2011.
- [26] McCrae, A.: Washington and Lee University geometric probability course notes on convex sets. <http://home.wlu.edu/~mcraea/GeometricProbabilityFolder/ApplicationsConvexSets/Problem16/Problem16.html>, 2011.
- [27] Dubins, L.E.: On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, vol. 79, pp. 497–516, 1957.
- [28] Brandon, J.: Fixed wing flight training: Manoeuvring forces. http://www.pilotfriend.com/training/flight_training/aero/man_force.htm, 2012.
- [29] United States Naval Air Training Command: Flight training instruction, Advanced Naval Flight Officer (T2C), CNATRA P-821 rev.08-06. 2006.
- [30] Scvortov, A.: Gaussian elimination. <http://compprog.wordpress.com/2007/12/11/gaussian-elimination/>, 2007.

- [31] Fraleigh, J.B. and Bearegard, R.A.: *Linear Algebra*. Addison-Wesley Publishing Company, 1995.
- [32] Nelder, J.A. and Mead, R.: A simplex method for function minimization. *The Computer Journal*, vol. 7, pp. 308–313, 1965.
- [33] McKinnon, K.I.M.: Convergence of the Nelder-Mead simplex method to a non-stationary point. *SIAM Journal on Optimization*, vol. 9, pp. 148–158, 1999.
- [34] Kirkpatrick, S., Gelatt, C.D. and Vecchi, M.: Optimization by Simulated Annealing. *Signal Propagation on Interconnects*, vol. 220, pp. 671–680, 1983.
- [35] Černý, V.: Thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, vol. 45, pp. 41–45, 1985.
- [36] Hestenes, M.R. and Stiefel, E.: Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, vol. 49(6), pp. 409–436, 1952.
- [37] Broyden, C.G.: The convergence of a class of double-rank minimization algorithms. *Journal of the Institute of Mathematics and Its Applications*, vol. 6, pp. 76–90, 1970.
- [38] Fletcher, R.: A new approach to variable metric algorithms. *The Computer Journal*, vol. 13(3), pp. 317–322, 1970.
- [39] Goldfarb, D.: A family of variable metric updates derived by variational means. *Mathematics of Computation*, vol. 24(109), pp. 23–26, 1970.
- [40] Shanno, D.F.: Conditioning of quasi-Newton methods for function minimization. *Mathematics of Computation*, vol. 24(111), pp. 657–664, 1970.
- [41] L. Knockaert, F. Ferranti, T.D.: Vector fitting vs. levenberg-marquardt : Some experiments. *13th IEEE Workshop on Signal Propagation on Interconnects*, pp. 1–4, 2009.
- [42] Levenberg, K.: A method for the solution of certain non-linear problems in least squares. *Quarterly of Applied Mathematics*, vol. 2, pp. 164–168, 1944.
- [43] Marquardt, D.W.: An algorithm for least squares estimation of nonlinear parameters. *Journal of Society for Industrial and Applied Mathematics*, vol. 11, pp. 431–441, 1963.
- [44] Fletcher, R.: A modified Marquardt subroutine for nonlinear least squares. *Atomic Energy Research Establishment, Harwell, England*, 1971.
- [45] Yu, H. and Wilamowski, B.M.: *Levenberg-Marquardt Training*, vol. 5 - Intelligent Systems of *Industrial Electronics Handbook*, chap. 12. 2nd edn. CRC Press, 2011.
- [46] Hagan, M.T. and Menhaj, M.: Training feedforward networks with the Marquardt algorithm. *IEEE Transactions on Neural Networks*, vol. 5, pp. 989–993, 1994.
- [47] Fletcher, R.: *Practical methods of Optimization*. Wiley New York, 1987.

- [48] Balda, M.: LMFnlsg - Solution of nonlinear least squares. <http://www.mathworks.com/matlabcentral/fileexchange/17534-lmfnlsg-solution-of-nonlinear-least-squares>, 2009.
- [49] Rosenbrock, H.H.: An automatic method for finding the greatest or least value of a function. *The Computer Journal*, vol. 3, pp. 175–184, 1960.
- [50] Igel, C., Toussaint, M. and Weishui, W.: *Trends and Applications in Constructive Approximation*, pp. 266–267. Birkhauser, 2005.
- [51] Intel Corporation: Intel®Math Kernel Library – LINPACK. <http://software.intel.com/en-us/articles/intel-math-kernel-library-linpack-download>, 2012.
- [52] National Meteorological Library and Archive: National Meteorological Library and Archive Fact sheet 6 – The Beaufort Scale. http://www.metoffice.gov.uk/media/pdf/4/4/Fact_Sheet_No._6_-_Beaufort_Scale.pdf, 2012.
- [53] Alvarado, K.: Los Angeles Airport reports 2011 passenger level up 4.7 percent over 2010; air cargo down 3.8 percent. *Los Angeles World Airports News Release*, January 2012.
- [54] Reed, E.: Traffic Alert and Collision Avoidance System (TCAS) 1981-1985. <http://www.bendixradiofoundation.com/documents/EarlyTCAS%20by%20E%20Reed.pdf>, 2012.
- [55] Los Angeles World Airports: Airport information – statistics. http://www.lawa.org/welcome_LAX.aspx?id=806, 2012.
- [56] Smith, B.: Concerning air traffic control safety in the Los Angeles area. Transcript of Statement before the House Committee on Science and Technology, Subcommittee on Transportation, Aviation and Materials, September 1981.